This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

THIS PAGE BLANK (USPTO)

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 13/00

A1

(11) International Publication Number: WO 98/21662

(43) International Publication Date: 22 May 1998 (22.05.98)

(21) International Application Number: PCT/US97/20232

(22) International Filing Date: 12 November 1997 (12.11.97)

(30) Priority Data:
60/030,906 14 November 1996 (14.11.96) US

(71) Applicant: MITSUBISHI ELECTRIC INFORMATION TECHNOLOGY CENTER AMERICA, INC. [US/US]; 201 Broadway, Cambridge, MA 02139 (US).

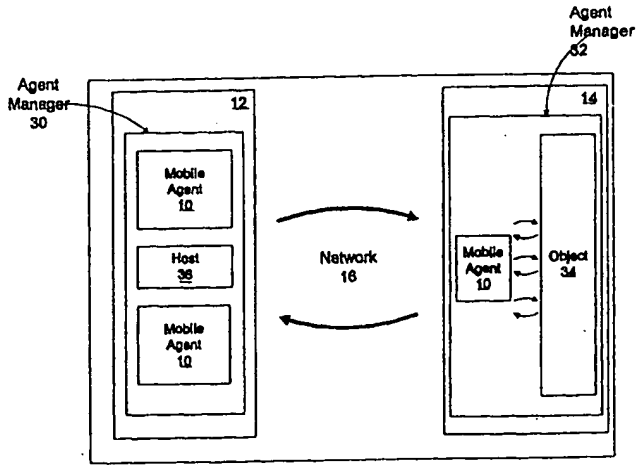
(72) Inventor: WALSH, Thomas, C.; 47 Tileston Street #2, Boston, MA 02113 (US).

(74) Agent: TENDLER, Robert, K.; 65 Atlantic Avenue, Boston, MA 02110 (US).

(81) Designated States: AU, CA, CN, IL, JP, KR, MX, NO, NZ, SG, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: ITINERARY BASED AGENT MOBILITY INCLUDING MOBILITY OF EXECUTABLE CODE



(57) Abstract

In accordance with the present invention, a mobile agent object (10) executes a first method (18) on a first computer (12), migrates from the first computer (12) to a second computer (14), and executes a second method (20) on the second computer (14). The first and second methods and first and second computers are designated in an itinerary (28). The agent includes both data (22) and executable code (24, 26) which are serialized for transmission from the first computer to the second computer as data. The data containing the agent is then deserialized in the second computer to regenerate the agent in the form of an object. The executable code portion of the agent can be supplemented with code from a home codebase located on another computer.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

TITLE OF THE INVENTION
ITINERARY BASED AGENT MOBILITY INCLUDING MOBILITY OF EXECUTABLE CODE

CROSS REFERENCE TO RELATED APPLICATIONS

A claim of priority is made to U.S. provisional patent application Serial Number 60/030,906, entitled USE AND COLLABORATION OF MOBILE AGENTS IN A COMPUTER NETWORK, filed November 14, 1996.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

Not applicable

BACKGROUND OF THE INVENTION

The present invention is generally related to network computing, and more particularly to mobile objects.

Mobile objects that are transmitted across a computer network are known. Mobile objects are comprised of code and data, both of which are transmitted across the computer network. Technologies such as client-server protocols, remote procedure call protocols, and distributed object technologies are employed to transmit mobile objects across the computer network. These technologies implement either a "push" model or a "pull" model, both of which have drawbacks.

In a "pull" model the code for an executing object is downloaded from a network source such as a web server. When a particular portion of code becomes necessary for operation, that portion of code is first sought on the local filesystem. If the local filesystem does not include that portion of the code, a network request is sent to another computer in the network in order to prompt transmission of that portion of code to the local computer. However, as different classes of sub-objects are created, a separate network request must

be sent to retrieve code for each class. In the case of a Hyper-Text Transfer Protocol ("HTTP") request, each request must re-establish a Transmission Control Protocol/Internet Protocol ("TCP/IP") socket connection to the web server. Thus, when downloading a large number of classes, a large number of connections must be established and a large amount of network overhead is incurred.

Another drawback associated with the pull model is that as an object travels, the code which enables the object to operate must be re-downloaded at each computer. More particularly, when the object arrives at each a destination computer, the object downloads any code which is required but not present on the local filesystem, even if that code was downloaded at the previous destination computer. Previously downloaded code is not cached. Network overhead is incurred as a consequence.

In a "push" model the code for an executing object is carried with the mobile object. Prior to launching the mobile object, all of the code that will be needed by the object is identified. The code is packaged with the agent and pushed around the network to each destination computer. While the push model reduces network requests in comparison with the pull model, network overhead is still incurred because of the relatively large amount of code that is pushed to every destination computer. In some cases the object will push code for classes that are no longer needed. For example, an agent may create an instance of a particular class of object only in very exceptional circumstances. In such a case it is inefficient for the object to push the code for this class.

Another limitation of known mobile objects is difficulty in examining and predicting destination information. Known mobile objects initiate travel by performing a subroutine call. The method is sometimes given a name such as "go" or "moveTo," and the caller is responsible for specifying a name or Uniform Resource Locator ("URL") indicating the destination computer for the mobile object. When the travel method is called, the execution of the mobile object is

halted and the mobile object is converted into a network transmittable form. On the destination computer the mobile object is restored from the network form and restarted. This restart can either occur on the instruction directly following the call to the travel method, or at the beginning of some known method.

BRIEF SUMMARY OF THE INVENTION

In order to provide efficient mobility of code for a mobile agent object ("agent"), a "mobile codebase" object is constructed. The mobile codebase travels with the agent and serves as a repository of code for classes that facilitate agent operation. When the agent is launched, the launcher specifies a list of related classes. These classes are retained in the mobile codebase. The launcher also specifies a Uniform Resource Locator ("URL") that points to a network source ("home codebase") from which code can be downloaded. If the agent is required to instantiate an object comprising code that is not in the mobile codebase, a network request is sent to the home codebase to retrieve that code. Code that is downloaded in this manner is retained in the mobile codebase.

The combination of the mobile codebase and home codebase improve agent operation and reduce network overhead. In a preferred embodiment an agent is launched with a mobile codebase that is preloaded with classes which are certain to be needed. Classes that are unlikely to be needed are stored in the home codebase. Thus the agent with mobile codebase and home codebase provide the desirable features of the push and pull models without the undesirable network overhead incurred by transmitting unneeded classes around the network or re-downloading code.

Agent travels are defined by an itinerary data structure. The itinerary is composed of a list of destinations, each of which includes: (1) the hostname of a destination computer on the network to which the agent should travel, and (2) the name of a method that the agent should

execute when the agent arrives at the destination computer. An agent is allowed to modify its itinerary. When an agent modifies its itinerary, the agent owner is automatically notified, thus allowing the owner to "follow" the agent wherever it may travel.

The itinerary provides advantages related to agent and network management. Inspection of the agent itinerary reveals where the agent has travelled and where the agent may travel in the future. Since the itinerary is a standard data structure, an observer can depend on it to be present. Further, since the itinerary is a separate data structure from the agent, the observer does not need special knowledge of the internal structure of the agent in order to examine the itinerary.

BRIEF DESCRIPTION OF THE DRAWING

The invention will be more fully understood from the following Detailed Description of the Invention, in conjunction with the Drawing, of which:

Fig. 1 is a block diagram which illustrates agent migration;

Fig. 2 is a block diagram which illustrates agent execution in conjunction with agent migration;

Fig. 3 is a block diagram of an agent;

Fig. 4 is a block diagram of agent interaction with agent managers on multiple computers in a network;

Fig. 5 is a flow chart which illustrates agent mobility;

Fig. 6 is a block diagram which illustrates agent operation according to an itinerary;

Fig. 7 is a block diagram which illustrates the agent runtime environment; and

Fig. 8 is a flow chart which illustrates agent deserialization.

DETAILED DESCRIPTION OF THE INVENTION

Referring to Fig. 1, an agent 10 comprises a mobile

object which can travel from a first computer 12 to a second computer 14 in a computer network 16. The agent is generated and stored in memory on the first computer for a specific purpose, and includes both data and executable code. The agent executes on the first computer and then migrates to the second computer by being transmitted across the network. The agent then resumes execution on the second computer. When the agent completes execution on the second computer, the agent acts in accordance with an agent itinerary. For example, the agent will migrate to another computer if the itinerary so indicates.

Referring now to Figs. 1 and 2, the agent executes a sequence of instructions during operation. In particular, a first method 18 is executed on the first computer 12 and a second method 20 is executed on the second computer 14. As a result, migration between computers is transparent from the perspective of the agent.

Referring to Fig. 4, agent mobility is preferably facilitated through use of JAVA agent manager Objects 32 and an Object Serialization facility (a product of Sun Microsystems which is known in the art). Object Serialization operates to serialize objects such that a representative stream of bytes, i.e., data, is generated therefrom. Object Serialization facilitates transmission of an agent across the network by serializing the agent 10 into a format that is suitable for network transmission. Serialization of the agent may also include serialization of sub-objects which are contained within member variables of the agent. In accordance with the present invention, the serialization process involves the serialization of executable code, data, and an itinerary as hereinafter discussed in greater detail. Once the agent has been serialized, a communication socket to the second computer is opened and the resulting stream of bytes is transmitted to the second computer along the socket.

The serialized agent is regenerated in the second computer 14 upon arrival. In particular, the stream of bytes transmitted along the socket is deserialized in the second

computer. Deserialization regenerates the agent in the form in which the agent existed prior to migration, i.e., an object. Following deserialization, the agent executes a predetermined method specified by its itinerary. The predetermined method may prompt interaction with another object 34 located on the second computer. When the method has been executed, the agent is serialized and transmitted to yet another computer, or possibly back to the first computer. The agent is therein deserialized upon arrival and proceeds to execute another predetermined method and interact with that computer. In the case of the agent migrating back to the first computer, such interaction could be with a host application 36 through which an information query was entered. The agent continues migrating across the network in this manner as execution requires. Eventually, the agent is terminated when execution of the itinerary is complete. It should be noted that the agent need not necessarily migrate to the initial host (first) computer after executing on a remote (second) computer. The agent could travel to multiple computers, and may not return to the initial host computer.

Referring to Fig. 3, each agent may include data 22, code 24, sub-objects 26 and an itinerary 28. The data portion 22 includes internal state information associated with the agent. The code portion 24 includes executable code associated with operation of the agent. The itinerary portion 28 specifies destinations for the agent. Sub-objects 26 include code of distinct objects contained within the agent. The agent carries the code for these sub-objects.

The code 24 portion of the agent 10 includes a mobile codebase and a reference pointer to a home codebase. The mobile codebase comprises executable code which is retained as part of the agent. When the agent migrates, the mobile codebase is transmitted along with the data, sub-objects and itinerary to the destination. The home codebase is executable code which resides on the computer which originally created and launched the agent or a suitable network server (such as a web server). The reference pointer

to the home codebase is a Uniform Resource Locator ("URL") that allows remote access to the home codebase from other computers in the network. If the agent requires code that is not present in the mobile codebase during execution, the agent retrieves the required code from the home codebase and retains such retrieved code in the mobile codebase.

Referring to Figs. 3 and 6, the itinerary 28 is a data structure which defines agent migration parameters. The itinerary is composed of a list of destinations and the name of a method associated with each such destination. In a preferred embodiment, each destination entry contains the Transmission Control Protocol/Internet Protocol ("TCP/IP") host name of a computer on the network to which the agent is designated to migrate. The method associated with the respective destination is invoked upon arrival at that destination. Hence, each destination represents a location and a task to be performed at the location.

Figs. 4 and 5 illustrate agent mobility. When the determination is made that the agent should migrate, the agent manager 30 examines the itinerary 28 to ascertain the next destination to which the agent 10 is designated to migrate. The agent manager 30 establishes a network connection to the agent manager 32 at the destination computer 14. The agent manager 30 passes the agent and related travel information to the destination agent manager 32. The agent manager 30 then deletes the local copy of the agent, terminates any execution threads which were being used by the agent, and performs other appropriate clean up activities.

From the perspective of the destination computer, the agent arrives at the destination computer as illustrated in step 38. As previously described, the agent arrives in the form of data which includes the code, sub-objects, itinerary and other data. Java/Object Serialization assists the de-serialization and reconstruction of the agent as depicted in step 40. The server then spawns a new thread in which the agent will execute as shown in step 42. A persistent local copy of the agent is then made as illustrated in step 44.

As depicted in step 46 the agent manager invokes the proper method of the agent for execution in the spawned thread at this node. The itinerary contains information indicating which method should be invoked at each designated destination computer. The agent then executes as illustrated in step 50.

When the agent has completed execution of the indicated method, another persistent copy of the agent is stored on the local disk as depicted in step 52 in order to prevent redundant execution in the event that the computer malfunctions before the agent migrates. The agent manager then examines the agent itinerary as illustrated in decision step 54. If the itinerary does not indicate any further destinations then agent execution is deemed complete as shown in step 56. However, if the itinerary indicates further destinations, then the agent manager determines whether the next destination is associated with another computer in decision step 58. If the next destination is not another computer, flow returns to step 44, thereby bypassing network traffic. However, if the next destination is associated with a different agent manager then a network connection to the indicated agent manager is established as depicted in step 60. The destination contains a string which indicates where to travel. The string contains either the name (host name) of a computer or a URL to a distributed object. The agent is then serialized as illustrated in Step 62 and subsequently transmitted to the next agent manager as depicted in Step 64.

Migration in accordance with the itinerary 28 is further illustrated in Fig. 6. The itinerary includes a listing including identification of each agent manager 66 to which the agent is configured to migrate and identification of a method 68 to execute at each respective agent manager. The agent migrates through the agent managers listed in the itinerary sequentially in the order in which the servers are listed, invoking each respective method upon arrival. Such method could be, for example, querying or updating a database on the destination computer.

Fig. 7 illustrates the agent runtime environment. Each

computer 70 on the network includes a Java Virtual Machine 72 running therein. A agent manager 74 runs on each respective Virtual Machine 72. The agent manager includes a plurality of threads 76, each thread having an agent 78 associated therewith. In particular, a new thread is formed when a new agent arrives at the server, and the agent executes on that thread. Security mechanisms prevent each agent from operating outside the scope of the associated thread. When agent execution completes, the agent migrates to the next destination identified in the itinerary.

In order for migration and execution to function properly, the agent manager modifies the mechanism by which Java loads classes. The Java standard library provides an object called a "ClassLoader" which can be subclassed by a programmer to modify the rules that Java follows to load classes. The ClassLoader class allows web browsers to download "applets" from a web server. The agent manager provides a specialized ClassLoader which allows the executable code for mobile agents to travel with the agent.

During a typical execution of a Java program, a Java interpreter loads classes as needed for execution from a logical file system on the local computer. However, agents become separated from the home computer and the home codebase during migration, i.e., agents are separated from the directory or directories on the filesystem of the home computer where the code is stored. Further, sub-objects sometimes need to be created as the agent executes, either as member variables or as temporary variables within the methods. These sub-objects could be instances of classes which are part of the standard Java packages or they could be instances of new classes written by the programmer of the agent. However, since the agent is executing on a different computer than the home computer following migration, the Java virtual machine may not be able to directly retrieve the code for the particular sub-object. To allow agents to properly load classes and construct objects when remote from the home computer, a special ClassLoader is employed by the agent manager whenever an agent attempts to construct an object.

The ClassLoader object first attempts to locate a class on the local file system of the computer. If the class is not on the local file system, the ClassLoader searches the mobile codebase. If such attempt fails, the ClassLoader sends a request back to the home machine of the agent. The agent's travel information contains a reference such as a URL which refers back to the home codebase on the home computer, and by accessing the URL the ClassLoader can retrieve classes from the home computer.

Agent deserialization is illustrated in Fig. 8. As depicted in step 80 the serialized agent arrives as a stream of data. Java Development Kit Object Serialization classes then call an agent manager related method to perform deserialization as shown in step 82. The method reads agent related objects from the object stream as depicted in step 84. In particular, the mobile codebase, itinerary and codebase URL are read by the method. A custom Classloader and ObjectInputStream are constructed as shown in step 86. Object Serialization then reads the agent and other objects from the input stream as depicted in step 88 and step 90, respectively. When the related objects are read, the ClassLoader retrieves the classes which comprise the agent as illustrated in step 92.

The new object's class is loaded once the classes are retrieved. As depicted in step 94, if the class is determined to have been previously loaded then the previously loaded class instance is employed (step 96) and the object is instantiated using the loaded class object as shown in step 98. However, if the class has not been previously loaded as determined in inquiry step 94, then inquiry is made as to whether the class is on the local class path of the destination computer as illustrated in decision step 100. If the class is present on the local machine then the class is loaded therefrom as shown in step 102 and the object is instantiated using the loaded class object as depicted in step 98. However, if the class is not found on a local class path as shown in step 100 then an inquiry is made as to whether the class is in the mobile codebase as shown in step

104. If the class is located in the mobile codebase then the class is loaded therefrom as shown in step 106 and the object is instantiated using the loaded class object as shown in step 98. However, if the class is not located in the mobile codebase in decision step 104 then the class is loaded by accessing the home codebase as shown in step 108. If the codebase pointer is an Internet type pointer such as a Hyper-Text Transfer Protocol ("HTTP") URL, as determined in inquiry step 110, then an HTTP request is sent to the local web server as shown in step 112 and the retrieved class is added to the Mobile CodeBase as depicted in step 114. However, if the codebase URL is not a HTTP URL then a remote method request is sent to a RemoteClassLoader object as shown in step 116 and the class is added to the mobile codebase as shown in step 114. In either case, the object is instantiated using the loaded class object as shown in step 98. If more objects exist in the stream to be retrieved, as determined in decision step 118, then another object is read from the input stream as illustrated in step 90. If there are no more objects to be retrieved, flow terminates.

Having described the preferred embodiments of the invention, other embodiments which incorporate concepts of the invention will now become apparent to one of skill in the art. Therefore, the invention should not be viewed as limited to the disclosed embodiments but rather should be viewed as limited only by the spirit and scope of the appended claims.

A program listing follows in the attached APPENDIX.

S PAGE BLANK (USPTO)

```
/*  $Header: /com/meitca/hsl/zonesjagents/shared/Agent.java 13    9/30/96 6:23p Walsh $
 *
 *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
 *  Information Technology America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Base class for mobile Agent
 *
 *      $Log: /com/meitca/hsl/zonesjagents/shared/Agent.java $
 *
 * 13    9/30/96 6:23p Walsh
 * clean up javadoc comments
 *
 * 12    9/29/96 4:09p Walsh
 * Give Agent access to AgentPackage information
 *
 * 11    9/28/96 6:57p Walsh
 * Comment out debug println
 *
 * 10    9/09/96 4:53p Walsh
 * Add prepareForTransport, completedTransport methods
 *
 * 9     9/04/96 3:18p Noemi
 * Added static initializer.  Moved functionality related to collaboration
 * to new CollaboratorAgent subclass.
 *
 * 8     8/30/96 4:30p Noemi
 * Changed constructors and finalizer to operate on AgentGroup references
 * (rather than AgentGroupImpl references).
 *
 * 7     8/28/96 2:45p Noemi
 * Removed SourceSafe conflict garbage.
 *
 * 6     8/28/96 2:20p Noemi
 * Added support for AgentGroups.
 *
 * 5     8/23/96 3:52p Walsh
 * Remove live method (superceded by ad-hoc invocation)
 *
 * 4     8/22/96 6:36p Walsh
 * Remove parameter to live method
 *
 * 3     8/13/96 2:23p Walsh
 * Add some comments & documentation
```

WO 98/21662

PCT/US97/20232

```
48   *
49   * 2     8/09/96 4:49p Walsh
50   */
51  package com.meitca.hsl.zonesjagents.shared;
52
53  import java.util.*;
54  import java.net.InetAddress;
55
56  import com.meitca.hsl.zonesjagents.conduit.*;
57
58
59  /**
60   * A base class representing a traveling agent program.  All user defined
61   * agents should be derived from the class Agent.  An agent
62   * travels from machine to machine as dictated by its Itinerary. At each stop,
63   * a specified method will be invoked.  After the method completes, the Agent
64   * will be sent to the next host on the itinerary.   To develop an agent, a
65   * programmer should derive a class from Agent.
66   * @see         ConduitServer
67   * @see     AgentGroup
68   * @author  Noemi Paciorek
69   * @author      Thomas Walsh
70   */
71  public abstract class Agent {
72
73          // Class variables
74          /** A counter used to generate the Agent's ID */
75          protected static int agentNum = 0;      // counter for agents started on a system
76
77          /** A unique ID representing the host the agent was initiated from*/
78          protected static String host;           // unique host ID
79
80          // Instance variables
81          /** A unique ID representing the Agent */
82          protected String agentID;               // unique agent ID
83
84          // A reference to the Agent's AgentPackage.  This reference will only be
85          // valid when the Agent is actually tavelling.  Prior to travel, this
86          // will be a null reference (for example the reference will be null
87          // when the Agents constructor is executing).
88          private AgentPackage itsPackage;
89
90          // Static initializer
91          static {
92              /*
93               * Try to obtain the host's name and IP address.  If this fails,
94               * construct a pseudo-random name and hope for the best!  (Of course,
```

```
95              * this should never happen.)
96              */
97             try {
98                 host = InetAddress.getLocalHost().toString();
99             } catch (Exception e) {
100                host = "__HOSTID__" + new Random().nextLong();
101            }
102        }
103
104        // Constructors
105        /** Constructs an Agent */
106    public Agent() {
107        /*
108         * Assign a unique agent ID.
109         */
110            try {
111                agentID = host + "/agent" + nextAgentNum();
112                //System.out.println("Agent ID = " + agentID);
113            } catch (Exception e) {
114                System.out.println("Agent constructor error: " + e.getMessage());
115                e.printStackTrace();
116
117            }
118
119                itsPackage = null;
120    }
121
122        // Class methods
123        private final synchronized int nextAgentNum() {
124            return ++agentNum;
125        }
126
127        // Instance methods
128        /**
129         * Retrieves the Agents's ID.  An AgentID is a unique String
130         * which identifies a particular instance of an Agent.
131         * @return      The Agent's ID.
132         */
133        public final String getAgentID() {
134            return agentID;
135        }
136
137        /**
138         * Converts an Agent to a String.  Implementation simply
139         * returns the AgentID.
140         * @return      A String representation of the Agent.
141         */
```

```
142         public final String toString() {
143             return getAgentID();
144         }
145
146         /**
147          * This is called by the ConduitServer immediatley prior to
148          * the agent being transported to its next destination.  Derived
149          * Agents can override this method to perform any final processing
150          * needed before transport
151          */
152         public void prepareForTransport() {
153         }
154
155         /**
156          * This is called by the ConduitServer when the Agent arrives at
157          * its new destination.  Derived Agents can override this method
158          * to perform any processing needed on arrival
159          */
160         public void completedTransport() {
161         }
162
163         /**
164          * Returns a reference to the Agents Itinerary.  The Itinerary
165          * is only available when the Agent is traveling.  It is not
166          * available during the Agent's constructor.  If the Itinerary
167          * is not available, this method returns null.
168          * @return The Agents Itinerary or null if the itinerary is
169          *                       not available
170          */
171         public Itinerary getItinerary() {
172                 if (itsPackage != null) {
173                         return itsPackage.getItinerary();
174                 } else {
175                         return null;
176                 }
177         }
178
179         /**
180          * Returns a URL pointing to the Agent's codebase on it home machine.
181          * This URL is used in conjunction with the AgentCodebase class to
182          * retrieve the bytecodes of the Agent and it related classes.
183          * The URL available when the Agent is traveling.  It is not
184          * available during the Agent's constructor.  If the URL
185          * is not available, this method returns null.
186          * @return The URL of the Agents codebase or null if the URL is
187          *                       not available
188          */
```

```
189         public String getHomeCodebaseURL() {
190                 if (itsPackage != null) {
191                         return itsPackage.getHomeCodebaseURL();
192                 } else {
193                         return null;
194                 }
195         }
196
197         /**
198          * This Method is called internally by the ConduitServer to
199          * provide  the Agents with a reference to its package.
200          */
201         public void setPackage(AgentPackage agentPackage) {
202                         itsPackage = agentPackage;
203         }
204
205 }
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/shared/AgentCodebase.java 3     11/11/96 5:25p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Class providing accessability to the home codebase of the agent
 *
 *      $Log: /com/meitca/hsl/zonesjagents/shared/AgentCodebase.java $
 *
 * 3     11/11/96 5:25p Billp
 * Made correction to company name
 *
 * 2     9/30/96 6:23p Walsh
 * clean up javadoc comments
 *
 * 1     9/28/96 6:35p Walsh
 * intial version
 *
 */
package com.meitca.hsl.zonesjagents.shared;

import java.net.*;
import java.io.*;
import sun.net.www.MeteredStream;

import com.meitca.hsl.zonesjagents.remoteloader.*;

/**
 * The AgentCodebase class provides accessability to the "home
 * codebase" of an agent.  As an agent travles, it may construct
 * an object whose bytecodes may not be installed on the machine
 * to which it has travelled.  In some cases, the agent may have
 * already constructed another instance of the same class or, the
 * user or programmer may have specified that this class was a
 * "related class" when launching the agent.  In both of these
 * cases, the bytecodes for the class should have already been
 * loaded into the Agent's MobileCodebase.  If the bytecodes for
 * the new object HAVE NOT already been loaded into the MobileCodebase,
 * the ConduitServer needs a mechanism for finding an retrieving
 * these bytecodes.  The AgentCodebase provides an mechanism through
 * which Agents can retrieve code from their home locations.
 * @see         Agent
 * @see         MobileCodebase
```

WO 98/21662 PCT/US97/20232

```
48   * @author      Thomas Walsh
49   */
50  public class AgentCodebase {
51          /**
52          * these constants identify the protocols that
53          * can be used to retrieve bytecodes from an
54          * agents codebase.
55          */
56          static final int UNKNOWN = -1;
57          static final int RMI = 0;
58          static final int HTTP = 1;
59          static final int FILE = 2;
60
61          /**
62           * The protocol to use to retrieve bytecodes from the
63           * AgentCodebase
64           */
65          int                                 itsProtocol;
66
67          /** An URL pointing back to the AgentCodebase */
68          String                      itsURL;
69
70          /**
71           * A ClassFileLoader object which may be used to
72           * retrieve agent bytecodes from the local filesystem
73           * (when a file: url is used to identify the codebase
74           */
75          ClassFileLoader itsClassFileLoader;
76
77          /**
78           * Constructs an AgentCodebase from  the given URL.
79           * @param       url An URL pointing to the AgentCodebase.  This
80           *                          URL can be;<br>
81           *                  <ul>
82           *                          <li>An HTTP url pointing to a location on a web server
83           *                                  (such as <b>http://host/directory</b>)
84           *                          <li>An RMI url pointing to a RemoteClassLoader
85           *                                  server (such as <b>rmi://host/RemoteClassLoader/directory</b>)
86           *                          <li>A FILE url pointing to a local directory
87           *                                  (such as <b>file:C:\directory</b>)
88           *                          <li>An empty string or null which indicates that the agents code
89           *                                  is on the CLASSPATH of this machine.
90           *                  </ul>
91       * @exception       MalformedURLException If the url passed in was invalid
92           */
93          public AgentCodebase(String url) throws MalformedURLException {
94                  itsProtocol = -1;
```

```
95                      itsURL = null;
96                      itsClassFileLoader= null;
97                      parseURL(url);
98
99                      if (itsProtocol == FILE)
100                             itsClassFileLoader = new ClassFileLoader();
101             }
102
103             /**
104              * Retrieve the bytecodes for the given class
105              * @param               classname The name of the class whose bytecodes to
106              *                                retrieve
107              * @return              The classes bytecodes in the form of a byte array
108              * @exception   ClassNotFoundException If the class could not be loaded
109              */
110             public byte[] retrieveCode(String classname) throws ClassNotFoundException {
111                     switch (itsProtocol) {
112                             case RMI:
113                                     return RemoteClassLoaderImpl.retrieveRemoteClass(itsURL, classname);
114                             case HTTP:
115                                     return retrieveHTTPClass(classname);
116                             case FILE:
117                                     return itsClassFileLoader.loadClassFileFromDirectory(itsURL, classname);
118                             default:
119                                     throw new ClassNotFoundException("Unsupported protocol");
120                     }
121             }
122
123             /* Used internally to parse URL passes into ctor */
124             private void parseURL(String url) throws MalformedURLException {
125                     if (url != null) {
126                             int sep = url.indexOf(':');
127
128                             if (sep != -1) {
129                                     String protocol = url.substring(0, sep);
130                                     if (protocol.equals("rmi")) {
131                                             itsProtocol = RMI;
132                                             itsURL = url;
133                                     } else if (protocol.equals("http")) {
134                                             itsProtocol = HTTP;
135                                             itsURL = url;
136                                             if (itsURL.endsWith("/")) {
137                                                     itsURL = itsURL.substring(0, itsURL.length()-1);
138                                             }
139                                     } else if (protocol.equals("file")) {
140                                             itsProtocol = FILE;
141
```

```
142                                     // strip off the file: portion so all we are left with is
143                                     // the directory spec of the codebase.
144                                     itsURL = url.substring(sep+1, url.length());
145
146                             } else {
147                                     // we will assume file since the user might be trying to specify a
148                                     // DOS style filename (ie C:\WINNT)
149                                     itsProtocol = FILE;
150                                     itsURL = url;
151                             }
152                     } else {
153                             // assume file
154                             itsProtocol = FILE;
155                             itsURL = url;
156                     }
157             } else {
158                     // the URL was null.  This means that the Agents code is located somewhere on
159                     // the class path of this machine.  We will set the protocol to FILE and leave
160                     // the url as null
161                     itsProtocol = FILE;
162             }
163     }
164
165     /* Used internally to retrieve .class files from a web server */
166     private byte[] retrieveHTTPClass(String classname) throws ClassNotFoundException {
167     try {
168                     // first we will try to load the agent using its fully qualified class
169                     // name.  If the HTTP URL was http://host/Agents and the class name
170                     // was test.TestAgent, we will build an URL like the following;
171                     // http://host/Agents/test/TestAgent.class
172         URL             classUrl = new URL(itsURL + "/" + classname.replace('.', '/') + ".class");
173
174         MeteredStream   stream = (MeteredStream)classUrl.getContent();
175         int             length = stream.available();
176         byte            data[] = new byte[length];
177
178         stream.read(data);
179         return data;
180     } catch (IOException ioerror) {
181                     // The fully qualified class name didn't work, so lets try to
182                     // strip off all of the pacakge names and search for just the
183                     // class name.  If the HTTP URL was http://host/Agents and the class name
184                     // was test.TestAgent, we will build an URL like the following;
185                     // http://host/Agents/TestAgent.class
186                     int lastSep = classname.lastIndexOf('.');
187                     if (lastSep != -1) {
188                             try {
```

WO 98/21662 PCT/US97/20232

```
AgentCodebase.java_1        Tue Nov 12 08:01:22 1996        5

                                    URL                     classUrl = new URL(itsURL + "/" + classname.substring(lastSe
    189
  + 1, classname.length()) + ".class");
    190
    191                             MeteredStream   stream = (MeteredStream)classUrl.getContent();
    192                             int             length = stream.available();
    193                             byte            data[] = new byte[length];
    194
    195                             stream.read(data);
    196                             return data;
    197                         } catch (Exception error) {
    198                             throw new ClassNotFoundException("Could not load class " + classname + "fro
  " + itsURL);
    199                     }
    200                 } else {
    201                     throw new ClassNotFoundException("Could not load class " + classname + "from " + its
  RL);
    202                 }
    203             } catch (Exception error) {
    204                 throw new ClassNotFoundException("Could not load class " + classname + "from " + itsURL);
    205             }
    206         }
    207     }
```

```
/*      $Header: /com/meitca/hsl/zonesjagents/shared/AgentConstants.java 2     11/11/96 5:25p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Constants used by Java Agents system.
 *
 *      $Log: /com/meitca/hsl/zonesjagents/shared/AgentConstants.java $
 *
 * 2     11/11/96 5:25p Billp
 * Made correction to company name
 *
 * 1     10/10/96 5:50p Walsh
 * initial version
 *
 */
package com.meitca.hsl.zonesjagents.shared;

import java.io.File;

/**
 * Class contains constants used within the Java Agent System.
 */
public class AgentConstants {
        /**
         * Contains the value of the "zones.home" property (which
         * points to the Zones installation directory)
         */
        public static String    ZONES_HOME;

        /**
         * Contains the value of the "zones.user.home" property (which
         * points to the Zones user specific directory)
         */
        public static String    ZONES_USER_HOME;

        /**
         * Contains the path of the images subdirectory of the Zones
         * installation directory.
         */
        public static String    IMAGE_DIR;

        /**
```

```
48          * Contains the path of the Agent subdirectory of the Zones
49          * installation directory.
50          */
51         public static String    AGENT_DIR;
52
53         /**
54          * Contains the path of the lib subdirectory of the Zones/Agents
55          * installation directory.
56          */
57         public static String    AGENT_LIB_DIR;
58
59         /**
60          * Contains the path of the Agent subdirectory of the Zones
61          * installation directory.
62          */
63         public static String    USER_AGENT_DIR;
64
65
66         /** Static Intializer is used to initialize some of the CONSTANTS */
67         static {
68                 ZONES_HOME = System.getProperty("zones.home", File.separator + "Zones");
69                 if (ZONES_HOME != null) {
70                         // terminate the directroy name with a separator if
71                         // it's not already terminated
72                         if (!ZONES_HOME.endsWith(File.separator))
73                                 ZONES_HOME += File.separator;
74
75                         IMAGE_DIR = ZONES_HOME + "images" + File.separator;
76                         AGENT_DIR = ZONES_HOME + "Agents" + File.separator;
77                         AGENT_LIB_DIR = AGENT_DIR + "lib" + File.separator;
78                 }
79
80                 ZONES_USER_HOME = System.getProperty("zones.user.home");
81                 if (ZONES_USER_HOME == null) {
82                                 // the zones.user.home Property was not defined.
83                                 // try the user.home Property
84                                 ZONES_USER_HOME = System.getProperty("user.home","");
85                                 if (ZONES_USER_HOME != null) {
86                                         // terminate the directroy name with a separator if
87                                         // it's not already terminated
88                                         if (!ZONES_USER_HOME.endsWith(File.separator))
89                                                 ZONES_USER_HOME += File.separator;
90
91                                         ZONES_USER_HOME += ".Zones";
92                                 }
93                         }
94
```

```
95                  if (ZONES_USER_HOME != null) {
96                          // terminate the directroy name with a separator if
97                          // it's not already terminated
98                          if (!ZONES_USER_HOME.endsWith(File.separator))
99                                  ZONES_USER_HOME += File.separator;
100
101                         USER_AGENT_DIR = ZONES_USER_HOME + "Agents" + File.separator;
102                 }
103         }
104 }
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/shared/ClassFileLoader.java 3     11/11/96 5:25p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Loads a ".class" file off of the local filesystem.
 *
 *      $Log: /com/meitca/hsl/zonesjagents/shared/ClassFileLoader.java $
 *
 * 3     11/11/96 5:25p Billp
 * Made correction to company name
 *
 * 2     9/30/96 6:23p Walsh
 * clean up javadoc comments
 *
 * 1     9/28/96 6:35p Walsh
 * intial version
 *
 */
package com.meitca.hsl.zonesjagents.shared;

import sun.tools.java.*;
import java.io.*;

/**
 * The ClassFileLoader class loads a Classes bytecodes off of the
 * local filesystem.  The user can either specify a particular directory
 * from which the bytecodes should be retieved, have the ClassFileLoader
 * search the system CLASSPATH or specify a custom Class Path which should
 * be searched for the bytecodes.
 * @author      Thomas Walsh
 */
public class ClassFileLoader {
        /**
         * The ClassPath which should be searched for ClassFiles.  This could
         * represent the system CLASSPATH, or a custom ClassPath specified by the
         * user.
         */
        ClassPath itsClassPath;

        /**
         * Constructs a ClassFileLoader object.  This constructor sets the
         * ClassFileLoader up so that it will search the system's CLASSPATH.
```

WO 98/21662 PCT/US97/20232

```
48           */
49          public ClassFileLoader() {
50                  itsClassPath = new ClassPath(System.getProperty("java.class.path"));
51          }
52
53          /**
54           * Constructs a ClassFileLoader object.  The pathstr parameter specifies
55           * a custom ClassPath from which bytecodes are to be retrieved.
56           * @param       pathstr  A custom ClassPath from which bytecodes are to be
57           *                       retrieved
58           */
59          public ClassFileLoader(String pathstr) {
60                  itsClassPath = new ClassPath(pathstr);
61          }
62
63          /**
64           * Loads a class from the objects class path.  Depending on which
65           * version of the constructor was used, this could mean that the
66           * system's CLASSPATH is searched or that a custom user-defined
67           * ClassPath is searched.
68           * @param               classname The fully qualified class name of the class to
69           *                              be loaded.
70           * @return              The bytecodes for the requested class in the form of a
71           *                              byte array
72           * @exception   ClassNotFoundException If the class could not be located on
73           *                              the ClassPath
74           */
75          public byte[] loadClassFileFromClassPath(String classname) throws ClassNotFoundException {
76                  String          filename = classname.replace('.', File.separatorChar) + ".class";
77                  ClassFile       file = itsClassPath.getFile(filename);
78
79                  if (file != null) {
80                          try {
81                                  long            length = file.length();
82                                  InputStream stream = file.getInputStream();
83                                  byte            data[] = new byte[(int)length];
84
85                                  stream.read(data);
86                                  return data;
87                          } catch (IOException e) {
88                                  throw new ClassNotFoundException(e.getMessage());
89                          }
90                  } else {
91                          throw new ClassNotFoundException("ClassFile not found on ClassPath: " + classname);
92                  }
93          }
94
```

```
1  /*      $Header: /com/meitca/hsl/zonesjagents/shared/JASProperties.java 4     11/11/96 5:25p Billp $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory.
4   *  Mitsubishi Electric Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *      Zones Agent Property File implementation.
11  *      $Log: /com/meitca/hsl/zonesjagents/shared/JASProperties.java $
12  *
13  * 4     11/11/96 5:25p Billp
14  * Made correction to company name
15  *
16  * 3     10/10/96 5:56p Walsh
17  * Make use of constants in AgentConstants class
18  *
19  * 2     10/10/96 2:30p Walsh
20  * Add default constructor
21  *
22  * 1     10/07/96 5:07p Walsh
23  * intial versions
24  *
25  */
26 package com.meitca.hsl.zonesjagents.shared;
27
28 import java.io.*;
29 import java.util.*;
30
31 /**
32   * The JASProperties supports the storage of
33   * user specific and server specific properties.
34   * <ul>
35   *                <li>User specific properties are stored in
36   *                        zones.user.home\Agents directory. If the
37   *                        zones.user.home System property is not defined,
38   *                        user properties are stored in the
39   *                        user.home\.Zones\Agents directory
40   *                <li>Server specific properties are stored in
41   *                        zones.home\Agents\Lib
42   * </ul>
43   * @see          Properties
44   * @author       Joe DiCelie
45   * @author       Tom Walsh
46   */
47 public class JASProperties extends PropertyFile {
```

```
141                                     return data;
142                             } else {
143                                     throw new ClassNotFoundException("Could not find " + classname +
144                                                                      " in directory " + dir);
145                             }
146                     }
147             } catch (IOException e) {
148                     throw new ClassNotFoundException("Could not load " + classname +
149                                                      ".  Error: " + e.getMessage());
150             }
151     }
152 }
```

```
95          /**
96           * Loads a  class from a particular directory.
97           * @param               dir The directory from which to retireve the class.  If
98           *                              this parameter is null or an empty string, the ClassPath
99           *                              is searched for the class
100          * @param               classname The fully qualified class name of the class to
101          *                              be loaded.
102          * @returns             The bytecodes for the requested class in the form of a
103          *                              byte array
104          * @exception   ClassNotFoundException If the class could not be located on
105          *                              the ClassPath
106          */
107         public byte[] loadClassFileFromDirectory(String dir, String classname) throws ClassNotFoundException {
108                 // if a null or an empty string was passed in for dir, fall back to
109                 // searching the class path for the class
110                 if ((dir == null) || (dir.length() == 0))
111                         return loadClassFileFromClassPath(classname);
112
113                 // place a sperator char on the end of the directory specification
114                 // if it does not already have one
115                 if (!dir.endsWith(File.separator)) {
116                         dir = dir + File.separator;
117                 }
118
119                 String  fullFilename = dir + classname.replace('.', File.separatorChar) + ".class";
120
121                 try {
122                         File            file = new File(fullFilename);
123
124                         if (file.exists()) {
125                                 long            length = file.length();
126                                 FileInputStream stream = new FileInputStream(file);
127                                 byte            data[] = new byte[(int)length];
128
129                                 stream.read(data);
130                                 return data;
131                         } else {
132                                 int lastSep = classname.lastIndexOf('.');
133                                 if (lastSep != -1) {
134                                         String filename = dir + classname.substring(lastSep + 1, classname.length()
+ ".class";
135                                         file = new File(filename);
136                                         long            length = file.length();
137                                         FileInputStream stream = new FileInputStream(file);
138                                         byte            data[] = new byte[(int)length];
139
140                                         stream.read(data);
```

```
48
49          /**
50           * This constant should be passed into constructor to
51           * specify that this object will contain <b>Server</b>
52           * specific properties.
53           */
54          public static final int SERVER = 0;
55
56          /**
57           * This constant should be passed into constructor to
58           * specify that this object will contain <b>User</b>
59           * specific properties.
60           */
61          public static final int USER = 1;
62
63          private static final String  PROP_FILE_SUFFIX = ".properties";
64          private static final String  DEFAULT_DESC = "Zones Property File";
65
66          /**
67        * Opens a Properties file and constructs a JASProperties object.
68           * @param               appName A unique name used to identify the application.
69           *                              The <i>appName</i> is used in constructing the name of the
70           *                              filename of the corresponding properties file.  The
71           *                              properties file will have a name of the form
72           *                              "appName.properties"
73           * @param               type Idientifies the type of properties file to open.
74           *                              This parameter can have the value <b>JASProperties.USER</b>
75           *                              or <b>JASProperties.SERVER</b>.  The <i>type</i> parameter
76           *                              controls where the proerties file is opened from.
77           *                              <ul>
78           *                                      <li>User specific properties are stored in
79           *                                              zones.user.home\Agents directory. If the
80           *                                              zones.user.home System property is not defined,
81           *                                              user properties are stored in the
82           *                                              user.home\.Zones\Agents directory
83           *                                      <li>Server specific properties are stored in
84           *                                              zones.home\Agents\Lib
85           *                              </ul>
86           * @param               description A description of this JASProperties object.
87           *                              The description will be written to the header of the
88           *                              Properties file if the save method is called
89           * @param               createIfNecessary If set to true, the JASProperties will
90           *                              create the Properties file if it does not already
91           *                              exist.  If set to false, then the constructor will
92           *                              throw a FileNotFoundException if the Properties file
93           *                              does not exists.
94           * @exception   FileNotFoundException If the Properties file could not
```

```
95          *                               be found and the createIfNecessary parameter is false.
96          * @exception   IOException If an error occurred while attempting to
97          *                               access the Properties file
98          * @exception   IllegalArgumentException If the value passes into the
99          *                               type argument was not valid.
100         */
101        public JASProperties(String appName, int type, String description,
102                                                        boolean createIfNecessary)
103                throws FileNotFoundException, IOException, IllegalArgumentException {
104
105                itsDescription = (description != null) ? description : DEFAULT_DESC;
106
107                // Open the default JAS security properties file.
108                itsFile = locatePropertiesFile(appName, type, createIfNecessary);
109
110                loadProperties();
111        }
112
113        /**
114         * Builds an "empty" Properties object.  This method can be used
115         * to construct a Properties object which returns only default values.
116         * This behavior may be desired if the Properties file can not be found
117         * and the program wishes to continue execution using its default values
118         * for preferences.
119         */
120        public JASProperties() {
121
122        }
123
124        /*
125         * Builds a prperties filename based on the appName and properties
126         * file type
127         */
128        private File locatePropertiesFile(String appName, int type,
129                                                boolean createIfNecessary)
130                throws IllegalArgumentException, FileNotFoundException, IOException {
131
132                String propDirName;
133                if (type == SERVER) {
134                        // We are accessing a server properties file.
135                        // Server specific properties are stored in
136                        // the directory zones.home\Agents\Lib.
137
138                        propDirName = AgentConstants.AGENT_LIB_DIR;
139                } else if (type == USER) {
140                        propDirName = AgentConstants.USER_AGENT_DIR;
141                } else {
```

```
142                     throw new IllegalArgumentException("Bad Properties File Type: " + type);
143             }
144
145
146             // See if the propperties directory exists
147             File propDir = new File(propDirName);
148             if (!propDir.exists()) {
149                     // The directory does not exist.
150                     // See if we should create it.
151                     if (createIfNecessary) {
152                             if (!propDir.mkdirs()) {
153                                     // We could not create the directory.
154                                     // Throw an IOException
155                                     throw new IOException("Could create properties directory "
156                                                                                     + propDir.toString()
157                             }
158                     } else {
159                             // The directory did not exists.
160                             // Throw a FileNotFoundException
161                             throw new FileNotFoundException("Properties directory "
162                                                     + propDir.toString() + " did not exists");
163                     }
164             }
165
166             // see if the properties file exists
167             File propFile = new File(propDirName + File.separator +
168                                                             appName + PROP_FILE_SUFFIX);
169             if (!propFile.exists()) {
170                     // The directory does not exist.
171                     // See if we should create it.
172                     if (createIfNecessary) {
173                             // create the file.
174                             FileOutputStream out = new FileOutputStream(propFile);
175                             PrintStream print = new PrintStream(out);
176                             print.println("#" + itsDescription);
177                             print.close();
178                             // That should have created the file
179                     } else {
180                             // The properties file did not exists.
181                             // Throw a FileNotFoundException
182                             throw new FileNotFoundException("Properties file "
183                                                     + propFile.toString() + " did not exists");
184                     }
185             }
186             return propFile;
187     }
```

WO 98/21662 PCT/US97/20232

JASProperties.java_1 Tue Nov 12 08:01:23 1996 5

```
188 )
189
190
191
192
```

```
/*      $Header: /ccm/meitca/hsl/zonesjagents/shared/PropertyFile.java 2     11/11/96 5:25p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *     All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Properties (Preference) File object.
 *
 *      $Log: /com/meitca/hsl/zonesjagents/shared/PropertyFile.java $
 *
 * 2     11/11/96 5:25p Billp
 * Made correction to company name
 *
 * 1     10/07/96 5:07p Walsh
 * intial versions
 *
 */
package com.meitca.hsl.zonesjagents.shared;

import java.io.*;
import java.util.*;

/**
 * The PropertyFile object extends the behavior of the Properties class
 * by associating the Properties with a disk file.  The PropertyFile
 * object automatically loads the Proeprties out of the file (using the
 * Properties.load method) and supplies a method for saving of any updated
 * Properties back to disk, called saveProperties.
 * @see         Properties
 * @author      Joe DiCelie
 * @author      Tom Walsh
 */
public class PropertyFile extends Properties {

        /** The name of the properties file corresponding to this Properties object */
        protected File itsFile;

        /** A description of this Properties object.   */
        protected String itsDescription;


        /**
         * Opens the specified  Properties file and constructs a
         * JASProperties object.
```

```
48          * @param               propertiesFileName Name of the Properties file to open
49          * @param               description A description of this JASProperties object.
50          *                              The description will be written to the header of the
51          *                              Properties file if the save method is called
52          * @exception   FileNotFoundException If the Properties file could not
53          *                              be found and the createIfNecessary parameter is false.
54          * @exception   IOException If an error occurred while attempting to
55          *                              access the Properties file
56          */
57         public PropertyFile(String propertiesFileName, String description)
58                                                         throws FileNotFoundException, IOException {
59
60                 itsFile = new File(propertiesFileName);
61                 itsDescription = description;
62                 loadProperties();
63         }
64
65         /**
66          * Protected constructor that can be used by derived classes.
67          */
68         protected PropertyFile() {
69                 itsFile = null;
70                 itsDescription = null;
71                 // The actual work of loading properties will be
72                 // done by derived class
73         }
74
75
76         // loads Properties from the Properties File
77         protected void loadProperties()
78                         throws FileNotFoundException, IOException {
79
80                 // open an input stream to the file specified
81                 FileInputStream in = new FileInputStream(itsFile);
82
83                 // load the properties
84                 load(in);
85         }
86
87         /**
88          * Saves the Properties to the corresponding Properties File
89          * @exception   FileNotFoundException if the Properties file could not be
90          *                              located
91          * @exception   IOException if an error occurs while accessing the
92          *                              properties file
93          */
94         public void saveProperties()
```

```
95                          throws FileNotFoundException, IOException {
96
97                  // open an input stream to the file specified
98                  FileOutputStream out = new FileOutputStream(itsFile);
99
100                 // load the properties
101                 save(out, itsDescription);
102         }
103
104         /**
105          * Refreshes the Properties object by re-reading the Properties File.  Any
106          * changes made to the Properties object will be lost.
107          * @exception   FileNotFoundException if the Properties file could not be
108          *                                    located
109          * @exception   IOException if an error occurs while accessing the
110          *                                    properties file
111          */
112         public void refreshProperties()
113                         throws FileNotFoundException, IOException {
114                 loadProperties();
115         }
116
117
118 }
119
120
121
122
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/bootstrap/AgentClassInfoGatherer.java 5     11/11/96 5:13p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  AgentClassInfoGatherer class peeks into an agents .class file to
 *  retrieve information needed for agent launching.
 *
 *      $Log: /com/meitca/hsl/zonesjagents/bootstrap/AgentClassInfoGatherer.java $
 *
 * 5     11/11/96 5:13p Billp
 * Made correction to company name.
 *
 * 4     9/30/96 6:35p Walsh
 * fix up javadoc comments
 *
 * 3     9/27/96 6:10p Dicelie
 * Minor change to loop which verified that the class to be launched was
 * subclasses from Agent.
 *
 * 2     9/10/96 11:46a Walsh
 * remove bogus imports that javac considers illegal
 *
 * 1     8/29/96 5:20p Walsh
 * intial version
 *
 */
package com.meitca.hsl.zonesjagents.bootstrap;

import java.io.*;
import com.meitca.hsl.zonesjagents.conduit.AgentSkeleton;

/**
 * This object peeks into the .class file of an agent to retrieve
 * some information needed for launching the agent.
 * AgentClassInfoGatherer verifies that the .class file indicated
 * does contain a class derived from Agent and then it verifies the
 * existence of the agent's invocation skeleton and finally loads
 * the skeleton class into memory and retreives a list of the
 * agents methods from the skeleton.
 * @see         Agent
 * @see         BootStrap
 * @see         AgentLaunchWizard
```

```
48    * @author      Thomas Walsh
49    */
50   public class AgentClassInfoGatherer {
51           /** The names of the agents methods */
52           String[]                              itsAgentsMethods;
53
54
55           /**
56            * Constructs a AgentClassInfoGatherer object.  Loads the agent
57            * and its skeleton.  Retrieves a list of the agent's methods.
58            * @param                 agentDir The directory containing the agent's
59            *                                 .class file.
60            * @param                 agentFilename The name of the agents's .class file.
61            * @exception   FileNotFoundException Thrown if the .class specified
62            *                                 could not be opened.
63            * @exception   IOException If an IO error occurs while teying to
64            *                                 access the agent's .class file.
65            * @exception   ClassFormatError If the agent's .class file does
66            *                                 not contain a valid class.
67            * @exception   NotAgentException If the agent's .class file does
68            *                                 not contain an agent.
69            * @exception   ClassNotFoundException If an error occured while
70            *                                 trying to access the agents .class file.
71            * @exception   NoAgentSkeletonException If the agent's invocation
72            *                                 skeleton could not be found.
73            */
74           public AgentClassInfoGatherer(String agentDir, String agentFilename)
75                           throws FileNotFoundException,  IOException,  ClassFormatError,
76                                   NotAgentException, ClassNotFoundException,
77                                   NoAgentSkeletonException {
78                   // let the loadAgentClassInfo method do all of the real work
79                   loadAgentClassInfo(agentDir, agentFilename);
80           }
81
82           /**
83            * Verifies the existence and correctness of the agent .class file
84            * and the agent's skeleton.  Retrieves a list of the agent's methods.
85            * Called internally by the constructor
86            * @param                 agentDir The directory containing the agent's
87            *                                 .class file.
88            * @param                 agentFilename The name of the agents's .class file.
89            * @exception   FileNotFoundException Thrown if the agent's .class
90            *                                 did not exist.
91            * @exception   IOException If an IO error occurs while teying to
92            *                                 access the agent's .class file.
93            * @exception   ClassFormatError If the agent's .class file does
94            *                                 not contain a valid class.
```

```
95          * @exception   NotAgentException If the agent's .class file does
96          *                                not contain an agent.
97          * @exception   ClassNotFoundException If an error occured while
98          *                                trying to access the agents .class file.
99          * @exception   NoAgentSkeletonException If the agent's invocation
100         *                                skeleton could not be found.
101         *
102         */
103        private void loadAgentClassInfo(String agentDir, String agentFilename)
104                throws FileNotFoundException,  IOException,  ClassFormatError ,
105                                NotAgentException, ClassNotFoundException,
106                                NoAgentSkeletonException {
107
108             // load the Agent class
109             BootStrapClassLoader classLoader = new BootStrapClassLoader(agentDir);
110             Class agent = classLoader.loadAgentClassFromFile(agentFilename, true);
111
112             // verify that it is actually an Agent
113             Class superClass = agent.getSuperclass();
114
115             // STABILITY: Any way to get these hard coded strings out of here?
116             String  superClassName;
117             do
118             {
119                 superClassName = superClass.getName();
120                     if (superClassName.equals("java.lang.Object")) {
121                             throw new NotAgentException(agentFilename + " does not contain an agent");
122                     }
123
124                     superClass = superClass.getSuperclass();
125
126         } while (!superClassName.equals("com.meitca.hsl.zonesjagents.shared.Agent"));
127
128             // ok, we have an agent....
129             // lets try to load the skeleton
130             try {
131                     // first lets strip the ".class" from the filename
132                     // STABILITY: Any way to get these hard coded strings out of here?
133                     String skelClassFileName = agentFilename.substring(0,
134                                                     agentFilename.length() - ".class".length());
135
136                     // append on the _Skel
137                     skelClassFileName = skelClassFileName + "_Skel.class";
138
139                     // now try to load the skeleton class
140                     Class skelClass = classLoader.loadAgentClassFromFile(skelClassFileName, true);
141
                        // finally retrieve the agents methods.
```

```
142                             AgentSkeleton skel = (AgentSkeleton)skelClass.newInstance();
143                             itsAgentsMethods = skel.getMethods();
144                     } catch (Exception e) {
145                             throw new ClassNotFoundException("Could not find skeleton for agent " + agentFilename);
146                     }
147             }
148
149             /**
150              * Retrieves a list of the agents methods.  This list is used
151              * by the Agent Launch GUI in order to present the user with
152              * a drop down list containing the eligible methods of the agent.
153              */
154             public String[] getAgentMethods() {
155                     return itsAgentsMethods;
156             }
157     }
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/bootstrap/AgentLaunchInformation.java 4     11/11/96 5:13p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  AgentLaunchInformation encapsulates the information needed to
 *  bootstrap the agent into the system.
 *
 *      $Log: /com/meitca/hsl/zonesjagents/bootstrap/AgentLaunchInformation.java $
 *
 * 4     11/11/96 5:13p Billp
 * Made correction to company name.
 *
 * 3     9/30/96 6:35p Walsh
 * fix up javadoc comments
 *
 * 2     9/11/96 4:46p Walsh
 * add itsAgent member
 *
 * 1     8/23/96 5:21p Walsh
 * intial version
 *
 */
package com.meitca.hsl.zonesjagents.bootstrap;

import com.meitca.hsl.zonesjagents.conduit.Itinerary;
import com.meitca.hsl.zonesjagents.shared.Agent;


/**
 * The AgentLaunchInformation class encapsulates the information
 * needed by the BootStrap class in order to launch and Agent.
 * This class is used internally by the AgentLaunchWizard and
 * the Agent command line launch tool.
 * @see         Agent
 * @see         BootStrap
 * @see         AgentLaunchWizard
 * @author      Thomas Walsh
 */
public class AgentLaunchInformation {
        /**
         * The directory (on the local host) containing the agent's
         * .class file as well as the .class files of any related
```

```
48          * classes.
49          */
50         public String           itsAgentDirectory;
51
52         /** The agent's .class file */
53         public String           itsAgentFile;
54
55         /**
56          * An array containing the filenames of any other classes
57          * that should be sent along with the agent.  The ConduitServer
58          * has the ability to communicate back to the agents home machine
59          * to retrieve classes needed at runtime (if the RemoteLoader server
60          * is running).  With this parameter a programmer can bypass this
61          * and send any needed classes up front.
62      */
63         public String[]         itsRelatedClasses;
64
65         /** The agents Itinerary */
66         public Itinerary        itsItinerary;
67
68         /**
69          * The agent itself.  This member on needs to be filled in if the
70          * caller wants to call a method on the agent other than the default
71          * constructor.  If this field is left blank, then the
72          * BootStrap.launchAgent method will constuct the Agent contained in
73          * the file itsAgentDirectory by calling its default constructor.
74          * This method gives the programmer the option of constructing an
75          * agent with a ctor other than the default.
76          */
77         public Agent            itsAgent;
78
79         /**
80          * A listing of the agents methods.  This member variable need not
81          * be filled in in order to launch the agent, but is used internally
82          * by the AgentLaunchWizard
83          */
84         public String[]         itsMethods;
85
86         /** Constructs an empty AgentLaunchInformation object */
87         public AgentLaunchInformation() {
88                 itsAgentDirectory = new String("");
89                 itsAgentFile = new String("");
90                 itsItinerary = new Itinerary();
91                 itsAgent = null;
92         }
93 }
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/bootstrap/AgentLaunchWizard.java 10    11/11/96 5:13p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  GUI tool for launcing agents into system
 *
 *      $Log: /com/meitca/hsl/zonesjagents/bootstrap/AgentLaunchWizard.java $
 *
 * 10    11/11/96 5:13p Billp
 * Made correction to company name.
 *
 * 9     10/10/96 5:57p Walsh
 * Make use of properties in AgentConstants class
 *
 * 8     10/08/96 10:21a Walsh
 * Fix name collision between bootstrap.StringResources and
 * util.StringResources
 *
 * 7     10/07/96 4:44p Walsh
 * Save name & location of last agent launched.  Restart Wizard pointing
 * at same agent.
 *
 * 6     10/07/96 3:55p Walsh
 * Set up code to look in zones.home\images for GIF file
 *
 * 5     9/30/96 6:35p Walsh
 * fix up javadoc comments
 *
 * 4     9/27/96 11:52a Walsh
 * print exception to stderr on LaunchException
 *
 * 3     9/09/96 3:46p Walsh
 * replace gjt.MessageDialog with com.meitca.hsl.util.InfoDialog.  Create
 * Security Manager in main
 *
 * 2     9/04/96 12:34p Walsh
 * Pop up dialog on any problems launching the agent
 *
 * 1     8/23/96 5:21p Walsh
 * intial version
 *
 */
```

WO 98/21662 PCT/US97/20232

```
48  package com.meitca.hsl.zonesjagents.bootstrap;
49
50  import java.awt.*;
51  import java.io.*;
52  import java.util.*;
53
54  import com.meitca.hsl.util.*;
55  import com.meitca.hsl.zonesjagents.security.*;
56  import com.meitca.hsl.zonesjagents.shared.*;
57
58  /**
59   * The AgentLaunchWizard provides a wizard-like GUI for launching
60   * Java Agents into the system
61   * @see          Agent
62   * @see          BootStrap
63   * @see          AgentLaunchInformation
64   * @see          Wizard
65   * @author       Thomas Walsh
66   */
67  public class AgentLaunchWizard extends Wizard {
68          /** The launch information */
69          AgentLaunchInformation  itsAgentInfo;
70
71          /** The filename of the GIF */
72          static final String             GIF_FILENAME = AgentConstants.IMAGE_DIR + "AgentLaunchWizard.gif";
73
74          /** Properties (Preference) object */
75          JASProperties                   itsProperties;
76
77          static final String             LAST_AGENT_DIR="zones.agent.bootstrap.LastAgentDir";
78          static final String             LAST_AGENT_FILE="zones.agent.bootstrap.LastAgentFile";
79          // Static initializer
80
81          /**
82           * Constructs an AgentLaunchWizard object.  Brings up the GUI and
83           * sets up the first panel.
84           */
85          public AgentLaunchWizard() {
86                  super(com.meitca.hsl.zonesjagents.bootstrap.StringResources.WIZARD_TITLE,
87                                  GIF_FILENAME);
88                  itsAgentInfo = new AgentLaunchInformation();
89
90                  try {
91                          itsProperties = new JASProperties(
92                                  com.meitca.hsl.zonesjagents.bootstrap.StringResources.WIZARD_APPNAME,
93                                  JASProperties.USER,
94                                  com.meitca.hsl.zonesjagents.bootstrap.StringResources.PROP_DESC,
```

```
95                              true);
96              } catch (Exception e) {
97                      itsProperties = null;
98              }
99
100             if (itsProperties != null) {
101                     itsAgentInfo.itsAgentDirectory = itsProperties.getProperty(LAST_AGENT_DIR, "");
102                     itsAgentInfo.itsAgentFile = itsProperties.getProperty(LAST_AGENT_FILE, "");
103             }
104
105             buildPanels();
106             firstPanel();
107
108             resize(570, 370);
109             show();
110     }
111
112     /**
113      * Builds the panels of the wizard.  The AgentLaunchWizard
114      * contains three panels 1) a panel for specifying the agent's .class
115      * file (names FindFilePanel), 2) a panel for specifying the related
116      * class files for sending with the agent (called RelatedFilesPanel) and
117      * 3) a panel for setting up the agents itinerary (called ItineraryPanel)
118      */
119     protected void buildPanels() {
120             FindFilePanel panel1 = new FindFilePanel(this, itsAgentInfo);
121             RelatedFilesPanel panel2 = new RelatedFilesPanel(this, itsAgentInfo);
122             ItineraryPanel panel3 = new ItineraryPanel(this, itsAgentInfo);
123     }
124
125     /**
126      * The finish method is called by the Wizard base class when the
127      * user has successfully filled all the necessary information
128      * into all the panels and has pressed the "Finish" button.  We now
129      * have all of the information we need so we'll try to launch the agent.
130      */
131     protected void finish() {
132             try {
133                     // launch the agent
134                     BootStrap.launchAgent(itsAgentInfo);
135
136                     if (itsProperties != null) {
137                             try {
138                                     itsProperties.put(LAST_AGENT_DIR, itsAgentInfo.itsAgentDirectory);
139                                     itsProperties.put(LAST_AGENT_FILE, itsAgentInfo.itsAgentFile);
140                                     itsProperties.saveProperties();
141                             } catch (Exception e) {
```

```
142                                     }
143                                 }
144
145                                 // shutdown.
146                                 cancel();
147                             } catch (LaunchException e) {
148                                 InfoDialog info = new InfoDialog(this,
149                                                 com.meitca.hsl.zonesjagents.bootstrap.StringResources.LAUNCH_ERROR_TITLE,
150                                                 com.meitca.hsl.zonesjagents.bootstrap.StringResources.LAUNCH_ERROR);
151                                 info.setBackground(Color.lightGray);
152                                 info.show();
153
154                                 System.err.println(e.getMessage());
155                                 e.printStackTrace();
156                             }
157                     }
158
159                     /**
160                      * main routine supplied so that AgentLaunchWizard can be initiated
161                      * from the command line
162                      */
163                     public static void main(String args[]) {
164                         // Create and install the security manager
165                         System.setSecurityManager(new InsecurityManager());
166
167                             AgentLaunchWizard gui = new AgentLaunchWizard();
168                     }
169
170 }
```

```
1  /*  $Header: /com/meitca/hsl/zonesjagents/bootstrap/AgentLaunchWizardPanel.java 4     11/11/96 5:13p Billp $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory,
4   *  Mitsubishi Electric Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  A subclass of WizardPanel for use in the AgentLaunchWizard
11  *
12  *      $Log: /com/meitca/hsl/zonesjagents/bootstrap/AgentLaunchWizardPanel.java $
13  *
14  * 4     11/11/96 5:13p Billp
15  * Made correction to company name.
16  *
17  * 3     9/30/96 6:35p Walsh
18  * fix up javadoc comments
19  *
20  * 2     9/10/96 11:46a Walsh
21  * remove bogus imports that javac considers illegal
22  *
23  * 1     8/29/96 5:21p Walsh
24  * intial version
25  *
26  */
27 package com.meitca.hsl.zonesjagents.bootstrap;
28
29 import java.awt.*;
30
31 import com.meitca.hsl.util.Wizard;
32 import com.meitca.hsl.util.WizardPanel;
33
34
35 /**
36  * AgentLaunchWizardPanel is a base class for the panels of the
37  * AgentLaunchWizard.  All of the panels in the launch wizard need
38  * shared access to the AgentLaunchInformation object describing the
39  * parameters of the launch.  This base class provides that
40  * access.
41  * @see         Agent
42  * @see         BootStrap
43  * @see         AgentLaunchInformation
44  * @see         AgentLaunchWizard
45  * @author      Thomas Walsh
46  */
47 public class AgentLaunchWizardPanel extends WizardPanel {
```

```
48          /** The launch parameters of the Wizard*/
49          protected AgentLaunchInformation        itsAgentInfo;
50
51          /**
52           * Constructs an AgentLaunchWizardPanel.
53           * @param       wizard The Wizard owning this panel (needed by super)
54           * @param       info The (shared) reference to the agent launch infomarion
55           */
56          public AgentLaunchWizardPanel(Wizard wizard, AgentLaunchInformation info) {
57                  super(wizard);
58                  itsAgentInfo = info;
59          }
60  }
61
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/bootstrap/BootStrap.java 11    11/11/96 5:13p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Module for boot strapping agents into the system.
 *
 *      $Log: /com/meitca/hsl/zonesjagents/bootstrap/BootStrap.java $
 *
 * 11    11/11/96 5:13p Billp
 * Made correction to company name.
 *
 * 10    10/01/96 10:39a Walsh
 * fix @exception javadoc tag
 *
 * 9     9/30/96 7:33p Walsh
 * add more javadoc
 *
 * 8     9/29/96 6:56p Walsh
 * Add new version of launchAgent
 *
 * 7     9/11/96 4:59p Walsh
 * allow launchInfo.itsRelatedClasses to be null
 *
 * 6     9/11/96 4:46p Walsh
 * if caller specified LaunchInfo.itsAgent, use that rather than
 * constructing a new one.
 *
 * 5     9/10/96 11:46a Walsh
 * remove bogus imports that javac considers illegal
 *
 * 4     9/09/96 3:46p Walsh
 * Move creation of SecurityManager to main to prevent error on multiple
 * calls to Launch
 *
 * 3     9/05/96 6:38p Walsh
 * Add a main method to handle command line agent launching
 *
 * 2     9/04/96 12:35p Walsh
 * Add some exception handling.  Let ConduitServer handle details of
 * launching agent.  Let RemoteLoader build up the home codebase URL
 *
 * 1     8/29/96 5:21p Walsh
```

```
48   * intial version
49   *
50   */
51  package com.meitca.hsl.zonesjagents.bootstrap;
52
53  import java.rmi.Naming;
54  import java.rmi.server.StubSecurityManager;
55  import java.rmi.server.MarshalOutputStream;
56  import java.net.InetAddress;
57  import java.util.*;
58  import java.io.*;
59
60  import com.meitca.hsl.zonesjagents.shared.*;
61  import com.meitca.hsl.zonesjagents.conduit.ConduitServer;
62  import com.meitca.hsl.zonesjagents.conduit.AgentPackage;
63  import com.meitca.hsl.zonesjagents.conduit.Itinerary;
64  import com.meitca.hsl.zonesjagents.conduit.MobileCodebase;
65  import com.meitca.hsl.zonesjagents.conduit.ConduitObjectOutputStream;
66  import com.meitca.hsl.zonesjagents.conduit.Destination;
67  import com.meitca.hsl.zonesjagents.conduit.ConduitServerImpl;
68  import com.meitca.hsl.zonesjagents.remoteloader.RemoteClassLoaderImpl;
69  import com.meitca.hsl.zonesjagents.security.InsecurityManager;
70
71
72
73  /**
74   * BootStrap provides a mechanism for boot strapping (launching) agents
75   * into the system.  The calling program must provide the launch parameters
76   * in the form of the AgentLaunchInformation object.  BootStrap takes this
77   * information and provides the necessary steps to launch the indicated agents.
78   * This includes building creating the MobileCodebase, loading any needed
79   * classes (including the agent) into the MobileCodebase, constructing the
80   * AgentPackage, contacting the intial ConduitServer and sending the agent.
81   * @see          Agent
82   * @see          AgentLaunchInformation
83   * @author       Thomas Walsh
84   */
85  public class BootStrap {
86          /**
87           * Launches an agent.
88           * @param       agent The agent to launch
89           * @param       itinerary The agent's itinerary
90           * @param       agentCodebaseURL An URL pointing to the codebase of the agent.
91           *                      This URL can take the following forms;
92           *                      <ul>
93           *
94           *                              <li>A file URL like <b>file:C:\agents</b> - which indicates that the
                                              agent's code is located on the local machine in the directory
```

```
95              *                                   <b>C:\agent</b>
96              *                                 <li>An HTTP URL like <b>http:/hostname/agents</b> - which indicates a
97              *                                   location on a web server from which the agents code can be
98              *                                   retrieved.  (This is very similar to the codebase modifier
99              *                                   which can be added to an HTML <b>&ltapplet&gt</b> tag).
100             *                                 <li>An RMI URL like <b>rmi:/hostname/RemoteClassLoader</b> pointing
101             *                                   to a RemoteClassLoader object which can retrieve the agents code.<br>
102             *                                   This RMI URL could also look like
103             *                                   <b>rmi:/hostname/RemoteClassLoader/C:\agents</b> which indicates that
104             *                                   the RemoteClassLoader on hostname should be used to retrieve
105             *                                   the classes and that the agents code is located in the <b>C:\agents</b>
106             *                                   directory.
107             *                           </ul>
108             * @param       relatedClasses An array of fully qualified classnames of classes which this agent makes
109             *                           use of.  These classes will be set up to travel with the agent.
110             * @exception   LaunchException is the agent could notbe launched.
111             */
112            public static void launchAgent(Agent agent, Itinerary itinerary, String agentCodebaseURL,
113                                                                                    String[] relatedClasses) throws LaunchExcept
on {
114                 try {
115                 MobileCodebase agentCode = new MobileCodebase();
116                 BootStrapClassLoader bootStrap = new BootStrapClassLoader(agentCodebaseURL,
117
agentCode);
118                 bootStrap.loadAgentClass(agent.getClass().getName(), false);
119
120                         if (relatedClasses != null ) {
121                                 for (int i=0; i<relatedClasses.length; i++)
122                                         Class related = bootStrap.loadAgentClass(relatedClasses[i], false);
123                         }
124
125                         String absoluteCodebaseURL;
126                         if (agentCodebaseURL!= null) {
127                                 if (!agentCodebaseURL.startsWith("rmi:") && !agentCodebaseURL.startsWith("http:"))
128                                         if (agentCodebaseURL.startsWith("file:")) {
129                                                 absoluteCodebaseURL = RemoteClassLoaderImpl.buildCodebaseURL(
130                                                                                                 InetAddress.getLoca
ost().getHostName(),
131                                                                                                 agentCodebaseURL.sub
tring(5, agentCodebaseURL.length()));
132                                         } else {
133                                                 absoluteCodebaseURL = RemoteClassLoaderImpl.buildCodebaseURL(
134                                                                                                 InetAddress.getLoca
ost().getHostName(),
135                                                                                                 agentCodebaseURL);
136                                         }
```

```
137                         } else {
138                                 absoluteCodebaseURL = agentCodebaseURL;
139                         }
140                 } else {
141                                 absoluteCodebaseURL = RemoteClassLoaderImpl.buildRmiURL(
142                                                                                 InetAddress.getLocalHost().g
tHostName());
143                 }
144
145             AgentPackage agentPackage = new AgentPackage(agent ,
                                                                                                        agen
146
Code,                                                                                                   iti
147
rary,                                                                                                   absol
148
uteCodebaseURL);
149
150
151                         // pass the agent along
152                         ConduitServerImpl.sendPackage(agentPackage);
153
154                 } catch (Exception error) {
155                         throw new LaunchException("LaunchException: " + error.getMessage());
156                 }
157         }
158
159         /**
160          * Launches an agent based on the given launch parameters
161          * @param       launchInfo The launch parameters
162          */
163     public static void launchAgent(AgentLaunchInformation launchInfo) throws LaunchException {
164
165         try {
166             MobileCodebase agentCode = new MobileCodebase();
167             BootStrapClassLoader bootStrap = new BootStrapClassLoader(launchInfo.itsAgentDirectory,
168
agentCode);
169             Class agentClass = bootStrap.loadAgentClassFromFile(launchInfo.itsAgentFile, true);
170
171                         if (launchInfo.itsRelatedClasses != null ) {
172                                 for (int i=0; i<launchInfo.itsRelatedClasses.length; i++)
173                                         Class related = bootStrap.loadAgentClassFromFile(launchInfo.itsRelatedClass
[i], true);
174                         }
175
176                         String codebaseURL = RemoteClassLoaderImpl.buildCodebaseURL(
177                                                                                 InetAddress.getLocalHost().getHostN
```

```
e(),                                                                launchInfo.itsAgentDirectory);
    178
    179
    180
    181             Agent agent;
    182                     if (launchInfo.itsAgent == null) {
    183                             agent = (Agent)agentClass.newInstance();
    184                     } else {
    185                             agent = launchInfo.itsAgent;
    186                     }
    187             AgentPackage agentPackage = new AgentPackage(agent ,                    agen
    188
Code,                                                                                       lau
    189
hInfo.itsItinerary,                                                                         code
    190
aseURL);
    191
    192
    193                     // pass the agent along
    194                     ConduitServerImpl.sendPackage(agentPackage);
    195
    196                 } catch (Exception error) {
    197                     throw new LaunchException("LaunchException: " + error.getMessage());
    198             }
    199     }
    200
    201         /**
    202          * This method gets invoked when a user tries to launch an Agent from
    203          * the command line.  The command line should look something like the
    204          * following;<br>
    205          * <pre>
    206          *
    207          * Usage: bootstrap [-d hostname,method] [-f relatedfile.class] [-h] agentfile.class
    208          *
    209          * </pre><br>
    210          * The user specifies the agent class file as the last parameter to
    211          * the command.  The destiantions for the Agent are specified with the
    212          * <b>-d</b> option.  A destination is specified by the hostname follows by a
    213          * comma (no space) followed by the name of the method to invoke.
    214          * This could look something like the following; <b>-d host1,agentMethod1</b>.
    215          * The user specifies related class files that should be sent with the
    216          * agent using the <b>-f</b> option.  Finally, the <b>-h</b> option brings up
    217          * some command line help.
    218          */
    219         public static void main(String args[]) {
    220             // Create and install the security manager
```

```
221             System.setSecurityManager(new InsecurityManager());
222
223                 // We start by doing some rather brute force parsing of the
224                 // command line.  The command line args are used to build up
225                 // an AgentLaunchInformation data structure.
226                 AgentLaunchInformation launchInfo = new AgentLaunchInformation();
227                 Vector destinationStrings = new Vector();
228                 Vector relatedFiles = new Vector();
229                 String agentFile = null;
230
231                 for (int i=0; i<args.length; i++ ) {
232                         if (args[i].equals("-d")) {
233                                 String destinationString = args[++i];
234                                 destinationStrings.addElement(destinationString);
235                                 continue;
236                         } else if (args[i].equals("-f")) {
237                                 String fileName = args[++i];
238                                 relatedFiles.addElement(fileName);
239                                 continue;
240                         } else if (args[i].equals("-h")) {
241                                 System.out.println(StringResources.USAGE);
242                                 return;
243                         } else {
244                                 if (args[i].charAt(0) == '-') {
245                                         System.out.println(StringResources.ILLEGAL_OPTN + args[i]);
246                                         System.out.println(StringResources.USAGE);
247                                         return;
248                                 } else {
249                                         agentFile = args[i];
250                                 }
251                         }
252                 }
253
254                 // first lets verify that we have a valid agent file.
255                 if (agentFile == null) {
256                         System.out.println(StringResources.NO_AGENT);
257                         System.out.println(StringResources.USAGE);
258                         return;
259                 }
260
261                 if (!agentFile.endsWith(".class")) {
262                         System.out.println(StringResources.ILLEGAL_AGENT + agentFile);
263                         System.out.println(StringResources.AGENT_HELP);
264                         return;
265                 }
266
267                 launchInfo.itsAgentDirectory = new String("");
```

```
launchInfo.itsAgentFile = agentFile;

for (int i=agentFile.length()-1; i>=0; i--) {
        if (agentFile.charAt(i) == File.separatorChar ) {
                launchInfo.itsAgentDirectory = agentFile.substring(0, i+1);
                launchInfo.itsAgentFile = agentFile.substring(i+1, agentFile.length());
                break;
        }
}

// Now let the AgentClassInfoGatherer verify the existence of the file,
// that the file contains a valid Java Agent and that the skeleton
// exists.
try {
        AgentClassInfoGatherer agentInfoGatherer = new AgentClassInfoGatherer(
                                launchInfo.itsAgentDirectory, launchInfo.itsAgentFile);

        launchInfo.itsMethods = agentInfoGatherer.getAgentMethods();
} catch (FileNotFoundException e1) {
        System.out.println(StringResources.ILLEGAL_AGENT + agentFile);
        System.out.println("\t" + StringResources.FILE_NOTFOUND);
        return;
} catch (IOException e1) {
        System.out.println(StringResources.ILLEGAL_AGENT + agentFile);
        System.out.println("\t" + StringResources.IOERROR);
        return;
} catch (ClassFormatError e1) {
        System.out.println(StringResources.ILLEGAL_AGENT + agentFile);
        System.out.println("\t" + StringResources.BAD_CLASS_FORMAT);
        return;
} catch (NotAgentException e1) {
        System.out.println(StringResources.ILLEGAL_AGENT + agentFile);
        System.out.println("\t" + StringResources.NOT_AGENT);
        return;
} catch (ClassNotFoundException e1) {
        System.out.println(StringResources.ILLEGAL_AGENT + agentFile);
        System.out.println("\t" + StringResources.BAD_CLASS_FORMAT);
        return;
} catch (NoAgentSkeletonException e1) {
        System.out.println(StringResources.ILLEGAL_AGENT + agentFile);
        System.out.println("\t" + StringResources.NO_SKELETON);
        return;
}

// Lets take a look at the related files
launchInfo.itsRelatedClasses = new String[relatedFiles.size()];
for (int i=0; i< relatedFiles.size(); i++) {
```

```
315                     String fileName = (String)relatedFiles.elementAt(i);
316                     if (fileName.endsWith(".class")) {
317                             launchInfo.itsRelatedClasses[i] = fileName;
318                     } else {
319                             System.out.println(StringResources.ILLEGAL_RELFILE + fileName);
320                             System.out.println(StringResources.RELFILE_HELP);
321                             return;
322                     }
323             }
324
325             // Finally, lets take a look at those destinations
326             if (destinationStrings.size() == 0) {
327                     System.out.println(StringResources.NO_DEST);
328                     return;
329             }
330
331             for (int i=0; i<destinationStrings.size(); i++) {
332                     String destinationString = (String)destinationStrings.elementAt(i);
333                     StringTokenizer tokenizer = new StringTokenizer(destinationString, ",");
334
335                     if (tokenizer.countTokens() != 2) {
336                             System.out.println(StringResources.ILLEGAL_DEST + destinationString);
337                             System.out.println(StringResources.USAGE);
338                     }
339
340                     String hostname = tokenizer.nextToken();
341                     String methodname = tokenizer.nextToken();
342
343                     // find out if the methodname actually point to one of the methods
344                     // in the Agent
345                     int methodId = -1;
346                     for(int j=0; j<launchInfo.itsMethods.length; j++) {
347                             if (launchInfo.itsMethods[j].equals(methodname)) {
348                                     methodId = j;
349                             }
350                     }
351
352                     if (methodId == -1) {
353                             System.out.println(StringResources.ILLEGAL_METHOD + methodname);
354                             System.out.println(StringResources.USAGE);
355                             return;
356                     } else {
357                             launchInfo.itsItinerary.addDestination(new Destination(hostname, methodId));
358                     }
359             }
360
361             // now that we have parsed the command line and built up the
```

```
Bootstrap.java_1        Tue Nov 12 07:44:47 1996        9

   362
   363                  // launchInfo, we will actually attempt to launch the agent.
   364                  try {
   365                      launchAgent(launchInfo);
   366                  } catch (LaunchException e) {
   367                      System.out.println(StringResources.LAUNCH_ERROR + ": " + e.getMessage());
   368                  e.printStackTrace();
   369                  }
   370              }
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/bootstrap/BootStrapClassLoader.java 3     11/11/96 5:13p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  ClassLoader used in agent bootstrap process.
 *
 *      $Log: /com/meitca/hsl/zonesjagents/bootstrap/BootStrapClassLoader.java $
 *
 * 3     11/11/96 5:13p Billp
 * Made correction to company name.
 *
 * 2     9/28/96 6:56p Walsh
 * Use AgentCodebase object to retrieve bytecodes.
 *
 * 1     8/29/96 5:21p Walsh
 * intial version
 *
 */
package com.meitca.hsl.zonesjagents.bootstrap;

import java.io.*;
import java.net.*;
import java.rmi.Naming;

import com.meitca.hsl.zonesjagents.conduit.*;
import com.meitca.hsl.zonesjagents.shared.*;
import com.meitca.hsl.zonesjagents.remoteloader.*;


/**
 * BootStrapClassLoader is used internally by the BootStrap
 * class to load agent related classes
 * @see         Agent
 * @see         BootStrap
 * @see         MobileCodebase
 * @author      Thomas Walsh
 */
public class BootStrapClassLoader extends ClassLoader {
        /** An URL pointing to the cobease from which class files should be retrieved */
        String itsHomeCodebaseURL;

        /** An object representing the home codebase of the agent */
```

```
48          AgentCodebase itsHomeCodebase;
49
50          /** The mobile code base into which to load classes */
51          MobileCodebase  itsMobileCodebase;
52
53          /**
54           * object which actually handles the dirty work or retrieving .class
55           * files for a given class.
56           */
57          ClassFileLoader                         itsClassFileLoader;
58
59
60          /**
61           * Constructs a BootStrapClassLoader.
62           * @param       url The directory from which this ClassLoader should
63           *                      retrieve .class files
64           */
65          public BootStrapClassLoader(String url) throws MalformedURLException {
66                  itsHomeCodebaseURL = url;
67                  itsHomeCodebase = new AgentCodebase(url);
68                  itsMobileCodebase = null;
69                  itsClassFileLoader = new ClassFileLoader();
70          }
71
72          /**
73           * Constructs a BootStrapClassLoader.  When this consturctor is
74           * called, the ClassLoader realizes that their is an associated
75           * MobileCodebase object.  As .class files are read from disk,
76           * they are also written into the MobileCodebase.  Once they are
77           * loaded into the MobileCodebase, they are ready to travel.
78           * @param       dir The directory from which this ClassLoader should
79           *                      retrieve .class files
80           * @param       codebase The associated MobileCodbase object
81           */
82          public BootStrapClassLoader(String url, MobileCodebase codebase) throws MalformedURLException {
83                  this(url);
84                  itsMobileCodebase = codebase;
85          }
86
87          /**
88           * Load a class.  The classes .class file must be in the directory
89           * that was specified in the constructor.  If a MobileCodebase
90           * was specified in the constructor, the class will also be loaded
91           * into that codebase.
92           * @param       name The class name
93           * @param       resolve True if resolveClass should be called
94           */
```

```
95      public Class loadAgentClass(String name, boolean resolve) throws ClassNotFoundException {
96
97              byte[] bytecodes = itsHomeCodebase.retrieveCode(name);
98
99              Class c = defineClass(bytecodes, 0, bytecodes.length);
100             if (resolve)
101                     resolveClass(c);
102
103             if (itsMobileCodebase != null)
104                     itsMobileCodebase.storeCode(c.getName(), bytecodes);
105
106             return c;
107     }
108
109         /**
110          * Load a class from the specified file.  The file must be in the
111          * directory that was specified in the constructor.  If a
112          * MobileCodebase was specified in the constructor, the class will
113          * also be loaded into that codebase.
114          * @param       name The class filename
115          * @param       resolve True if resolveClass should be called
116          */
117         //STABILITY: Integrate this method with loadAgentClass
118         public Class loadAgentClassFromFile(String fileName, boolean resolve)
119                 throws FileNotFoundException,  IOException,  ClassNotFoundException , ClassFormatError {
120
121         File            file = new File(itsHomeCodebaseURL, fileName);
122         long            length = file.length();
123         FileInputStream stream = new FileInputStream(file);
124         byte            data[] = new byte[(int)length];
125
126         stream.read(data);
127         Class c = defineClass(data, 0, (int)length);
128         if (resolve) {
129                         resolveClass(c);
130                 }
131
132                 if (itsMobileCodebase != null)
133                         itsMobileCodebase.storeCode(c.getName(), data);
134         return c;
135     }
136
137         /**
138          * The java.lang.ClassLoader version of loadClass.
139          * First looks for a standard system class, and then looks for an
140          * agent class in the directory that was specified in the constructor.
141          * If the class is an Agent related class, then its bytecodes are
```

WO 98/21662 PCT/US97/20232

```
142             * loaded into the MobileCodebase
143             * @param       name The class name
144             * @param       resolve True if resolveClass should be called
145             */
146         protected Class loadClass(String name, boolean resolve) throws ClassNotFoundException {
147                 try {
148                         Class c = findSystemClass(name);
149                         if (resolve) {
150                                 resolveClass(c);
151                         }
152                         return c;
153                 } catch (ClassNotFoundException e1) {
154                         try {
155                                 return loadAgentClass(name, resolve);
156                         } catch (Exception e2) {
157                                 throw new ClassNotFoundException("Could not load " + name);
158                         }
159                 }
160         }
161 }
```

```
/*      $Header: /com/meitca/hsl/zonesjagents/bootstrap/FindFilePanel.java 5     11/11/96 5:13p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  The FindFilePanel of the AgentLaunchWizard
 *
 *      $Log: /com/meitca/hsl/zonesjagents/bootstrap/FindFilePanel.java $
 *
 * 5     11/11/96 5:13p Billp
 * Made correction to company name.
 *
 * 4     10/07/96 4:44p Walsh
 * Save name & location of last agent launched.  Restart Wizard pointing
 * at same agent.
 *
 * 3     9/10/96 11:46a Walsh
 * remove bogus imports that javac considers illegal
 *
 * 2     9/09/96 3:45p Walsh
 * replace gjt.MessageDialog w/ com.meitca.hsl.util.InfoDialog
 *
 * 1     8/29/96 5:21p Walsh
 * intial version
 *
 */
package com.meitca.hsl.zonesjagents.bootstrap;

import java.awt.*;
import java.io.*;
import gjt.Util;

import com.meitca.hsl.util.Wizard;
import com.meitca.hsl.util.WizardPanel;
import com.meitca.hsl.util.InfoDialog;

/**
 * the first panel of the AgentLaunchWizard.  This panel allows the user
 * to specify the filename of the .class file containing the agent.
 * @see         Agent
 * @see         AgentLaunchWizardPanel
 * @see         AgentLaunchWizard
 * @author      Thomas Walsh
```

```
48   */
49  public class FindFilePanel extends AgentLaunchWizardPanel {
50          /** The edit field */
51          TextField                       itsFileBrowseField;
52
53          /** The "Browse..." button */
54          Button                          itsFileBrowseButton;
55
56          /**
57           * Construct the FindFilePanel and all of its components.
58           * @param       wizard The wizard to which this panel belongs (needed by
59           *                      super)
60           * @param       wiinfozard The AgentLaunchInformation descringing this
61           *                      agent launch (needed by super)
62           */
63          public FindFilePanel(Wizard wizard, AgentLaunchInformation info) {
64                  super(wizard, info);
65                  buildPanel();
66          }
67
68          /**
69           * Verify that the information entered into the panel is valid.  Called
70           * after the "Next >" button is pressed.
71           *
72           * The method validates the following things...
73           *              1) A class file name is specified
74           *              2) The file name ends with ".class"
75           *              3) The file exists
76           *              4) The file contains valid Java bytecodes
77           *              5) The file contains a class derived from Agent
78           *              6) An invocation skeleton is available.
79           * @returns     true if the panel contains valid info, false if not
80           */
81          public boolean validatePanel() {
82                  // we want to break the full filename up into the directory portion
83                  // and the filename portion.  Find the last occurrence of the
84                  // directory seperator character
85                  String fullName = itsFileBrowseField.getText();
86
87                  // Verify that someone typed something into the edit control and that
88                  // what they typed in ends with .class
89                  if (fullName.length() == 0) {
90                          InfoDialog info = new InfoDialog(itsWizard,
91                                                           StringResources.LAUNCH_ERROR_TITLE,
92                                                           StringResources.NO_FILENAME);
93                          info.setBackground(Color.lightGray);
94                          info.show();
```

```
95                      return false;
96              }
97
98              if (!fullName.endsWith(".class")) {
99                      InfoDialog info = new InfoDialog(itsWizard,
100                                                       StringResources.LAUNCH_ERROR_TITLE,
101                                                       StringResources.ENDSWITH_CLASS);
102                     info.setBackground(Color.lightGray);
103                     info.show();
104                     return false;
105             }
106
107             itsAgentInfo.itsAgentDirectory = new String("");
108             itsAgentInfo.itsAgentDirectory = fullName;
109
110             for (int i=fullName.length()-1; i>=0; i--) {
111                     if (fullName.charAt(i) == File.separatorChar ) {
112                             itsAgentInfo.itsAgentDirectory = fullName.substring(0, i+1);
113                             itsAgentInfo.itsAgentFile = fullName.substring(i+1, fullName.length());
114                             break;
115                     }
116             }
117
118             // Now let the AgentClassInfoGatherer verify the existence of the file,
119             // that the file contains a valid Java Agent and that the skeleton
120             // exists.
121             try {
122                     AgentClassInfoGatherer agentInfoGatherer = new AgentClassInfoGatherer(
123                                                      itsAgentInfo.itsAgentDirectory, itsAgentInfo.itsAgentFile);
124
125                     itsAgentInfo.itsMethods = agentInfoGatherer.getAgentMethods();
126                     return true;
127             } catch (FileNotFoundException e1) {
128                     InfoDialog info = new InfoDialog(itsWizard,
129                                                      StringResources.LAUNCH_ERROR_TITLE,
130                                                      StringResources.FILE_NOTFOUND);
131                     info.setBackground(Color.lightGray);
132                     info.show();
133                     return false;
134             } catch (IOException e1) {
135                     InfoDialog info = new InfoDialog(itsWizard,
136                                                      StringResources.LAUNCH_ERROR_TITLE,
137                                                      StringResources.IOERROR);
138                     info.setBackground(Color.lightGray);
139                     info.show();
140                     return false;
141             } catch (ClassFormatError e1) {
```

```
142                         InfoDialog info = new InfoDialog(itsWizard,
143                                                 StringResources.LAUNCH_ERROR_TITLE,
144                                                 StringResources.BAD_CLASS_FORMAT);
145                         info.setBackground(Color.lightGray);
146                         info.show();
147                         return false;
148                 } catch (NotAgentException e1) {
149                         InfoDialog info = new InfoDialog(itsWizard,
150                                                 StringResources.LAUNCH_ERROR_TITLE,
151                                                 StringResources.NOT_AGENT);
152                         info.setBackground(Color.lightGray);
153                         info.show();
154                         return false;
155                 } catch (ClassNotFoundException e1) {
156                         InfoDialog info = new InfoDialog(itsWizard,
157                                                 StringResources.LAUNCH_ERROR_TITLE,
158                                                 StringResources.BAD_CLASS_FORMAT);
159                         info.setBackground(Color.lightGray);
160                         info.show();
161                         return false;
162                 } catch (NoAgentSkeletonException e1) {
163                         InfoDialog info = new InfoDialog(itsWizard,
164                                                 StringResources.LAUNCH_ERROR_TITLE,
165                                                 StringResources.NO_SKELETON);
166                         info.setBackground(Color.lightGray);
167                         info.show();
168                         return false;
169                 }
170         }
171
172         /**
173          * Builds the GUI Components for this panel
174          */
175         private void buildPanel() {
176                 setLayout(new GridBagLayout());
177
178                 Label label1 = new Label(StringResources.FF_TEXT1, Label.LEFT);
179                 constrain(this, label1,
180                                         0, 0, 1, 1,
181                                         GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0
182                                         0,0,0,0);
183
184                 Label label2 = new Label(StringResources.FF_TEXT2, Label.LEFT);
185                 constrain(this, label2,
186                                         0, 1, 1, 1,
187                                         GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0
188                                         0,0,0,0);
```

```
189
190                 Label label3 = new Label(StringResources.FF_TEXT3, Label.LEFT);
191                 constrain(this, label3,
192                                         0, 2, 1, 1,
193                                         GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
194                                         0,0,0,0);
195
196                 Label label4 = new Label(StringResources.FF_TEXT4, Label.LEFT);
197                 constrain(this, label4,
198                                         0, 3, 1, 1,
199                                         GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
200                                         0,0,0,0);
201
202                 Panel browsePanel = new Panel();
203                 constrain(this, browsePanel,
204                                         0, 4, 1, 1,
205                                         GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 1.0, 1.0,
206                                         0,0,0,0);
207
208                 browsePanel.setLayout(new GridBagLayout());
209                 Label findFileLabel= new Label(StringResources.FF_FIELD_LABEL, Label.LEFT);
210                 constrain(browsePanel, findFileLabel,
211                                         1, 0, 1, 1,
212                                         GridBagConstraints.NONE, GridBagConstraints.NORTHWEST, 0.0, 0.0,
213                                         0,0,0,0);
214
215                 itsFileBrowseField = new TextField(25);
216                 constrain(browsePanel, itsFileBrowseField,
217                                         2, 0, 1, 1,
218                                         GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 1.0, 1.0,
219                                         0,0,0,0);
220
221                 itsFileBrowseButton = new Button(StringResources.FF_BROWSE_BUTTON);
222                 constrain(browsePanel, itsFileBrowseButton,
223                                         3, 0, 1, 1,
224                                         GridBagConstraints.NONE, GridBagConstraints.NORTHWEST, 0.0, 0.0,
225                                         0,0,0,0);
226
227         }
228
229         /** Derived from Component.action */
230         public boolean action(Event event, Object what){
231                 if (event.target == itsFileBrowseButton) {
232             doBrowse();
233                         return true;
234                 }
235                 return false;
```

```
236         }
237
238         /**
239          * Called in response to selecting the Panel's "Browse... button.  Brings up
240          * a Open File Dialog.
241          */
242         private void doBrowse() {
243                 FileDialog browse = new FileDialog(Util.getFrame(this), StringResources.BROWSE_DLG_TITLE);
244                 browse.setFile("*.class");
245
246                 if (itsAgentInfo.itsAgentDirectory != null) {
247                         if (itsAgentInfo.itsAgentDirectory.length() != 0) {
248                                 browse.setDirectory(itsAgentInfo.itsAgentDirectory);
249                         }
250                 }
251
252                 browse.show();
253
254                 String fullFileName = new String("");
255
256                 if (browse.getDirectory() != null) {
257                         if (browse.getDirectory().length() != 0) {
258                                 fullFileName = browse.getDirectory();
259                         }
260                 }
261
262                 if (browse.getFile() != null) {
263                         if (browse.getFile().length() != 0) {
264                                 fullFileName = fullFileName + browse.getFile();
265                         }
266                 }
267
268                 if (fullFileName.length() != 0)
269                         itsFileBrowseField.setText(fullFileName);
270         }
271
272         /**
273          * Initializes the Panel for display
274          */
275         public void displayPanel() {
276                 String fullFileName = "";
277
278                 if (itsAgentInfo.itsAgentDirectory != null) {
279                         if (itsAgentInfo.itsAgentDirectory.length() != 0) {
280                                 fullFileName = itsAgentInfo.itsAgentDirectory;
281
282                                 if (!fullFileName.endsWith(File.separator)) {
```

```
283                                     fullFileName += File.separator;
284                                 }
285                             }
286                 }
287
288
289                 if (itsAgentInfo.itsAgentFile != null) {
290                         if (itsAgentInfo.itsAgentFile.length() != 0) {
291                                 fullFileName += itsAgentInfo.itsAgentFile;
292                         }
293                 }
294
295                 if (fullFileName.length() != 0)
296                         itsFileBrowseField.setText(fullFileName);
297         }
298
299
300 }
```

```
/*      $Header: /com/meitca/hsl/zonesjagents/bootstrap/ItineraryGridDataModel.java 3     11/11/96 5:13p
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  A Data Model class for use in conjunction with the Rogue Wave
 *      Grig class
 *
 *      $Log: /com/meitca/hsl/zonesjagents/bootstrap/ItineraryGridDataModel.java $
 *
 * 3     11/11/96 5:13p Billp
 * Made correction to company name.
 *
 * 2     9/04/96 12:36p Walsh
 * Propogate changes to Destination terminology
 *
 * 1     8/29/96 5:21p Walsh
 * intial version
 *
 */
package com.meitca.hsl.zonesjagents.bootstrap;

import com.roguewave.widgets.*;
import com.roguewave.widgets.grid.*;
import java.util.Vector;
import java.util.Enumeration;

import com.meitca.hsl.zonesjagents.conduit.Destination;
import com.meitca.hsl.zonesjagents.conduit.Itinerary;

/**
 * Implements a the necessary com.rogue.widgets.grid.* interfaces to
 * support the display of a Grid object.  This class is tied to the
 * Grid control on the ItineraryPanel of the AgentLaunchWizard.  The
 * grid is used to specify the Itinerary of the Agent to launch
 * @see         Itinerary
 * @see         ItineraryPanel
 * @see         AgentLaunchWizard
 * @author      Thomas Walsh
 */
public class ItineraryGridDataModel extends SelectionGridModel {
        /** The rows displayed in the grid */
        Vector                  itsRows;
```

```
48
49          /** The ItineraryPanel this object is tied to. */
50          ItineraryPanel  itsPanel;
51
52          /**
53           * Construct a ItineraryGridDataModel
54           * @param       panel   The ItineraryPanel this object is tied to.
55           */
56      public ItineraryGridDataModel(ItineraryPanel panel) {
57                  itsRows = new Vector();
58                  itsPanel = panel;
59      }
60
61          /** derived from SelectionGridModel.fetchString */
62      public String fetchString(int row, int column) {
63                  try {
64                          // vectors are zero based...Grids are one based, so we need to
65                          // subtract one
66                          Destination aRow = (Destination)itsRows.elementAt(row-1);
67                          switch (column) {
68                                  case 1:
69                                          return aRow.getDestinationHost();
70                                  default:
71                                          return "";
72                          }
73                  } catch (ArrayIndexOutOfBoundsException e){
74                          return "";
75                  }
76      }
77
78          /** derived from SelectionGridModel.fetchInteger */
79      public int fetchInteger (int row, int column) {
80                  try {
81                          // vectors are zero based...Grids are one based, so we need to
82                          // subtract one
83                          Destination aRow = (Destination)itsRows.elementAt(row-1);
84                          switch (column) {
85                                  case 2:
86                                          return aRow.getMethodID() + 1;
87                                  default :
88                                          return 0;
89                          }
90                  } catch (ArrayIndexOutOfBoundsException e){
91                          return 0;
92                  }
93      }
94
```

```
95          /** derived from SelectionGridModel.update */
96      public boolean update (int row, int column, String value) {
97              Destination     aRow;
98
99              try {
100                     // vectors are zero based...Grids are one based, so we need to
101                     // subtract one
102                     aRow = (Destination)itsRows.elementAt(row-1);
103             } catch (ArrayIndexOutOfBoundsException e){
104                     aRow = new Destination();
105                     itsRows.insertElementAt(aRow, row-1);
106
107                     // a new destination has been added to the Itinerary.  Lets add
108                     // a new blank row to the Grid so that the user has space to type any
109                     // more destinations
110                     itsPanel.addRow();
111             }
112
113             switch (column)  {
114                     case 1:
115                             aRow.setDestinationHost(value);
116                             break;
117                     default:
118                             break;
119             }
120             return true;
121         }
122
123         /** derived from SelectionGridModel.update */
124         public boolean update (int row, int column, int value) {
125                 Destination             aRow;
126
127                 try {
128                         // vectors are zero based...Grids are one based, so we need to
129                         // subtract one
130                         aRow = (Destination)itsRows.elementAt(row-1);
131                 } catch (ArrayIndexOutOfBoundsException e){
132                         aRow = new Destination();
133                         itsRows.insertElementAt(aRow, row-1);
134
135                         // a new destination has been added to the Itinerary.  Lets add
136                         // a new blank row to the Grid so that the user has space to type any
137                         // more destinations
138                         itsPanel.addRow();
139                 }
140
141                 switch (column)  {
```

WO 98/21662 PCT/US97/20232

```
142                             case 2:
143                                     aRow.setMethodID(value - 1);
144                                     break;
145                             default:
146                                     break;
147                     }
148                     return true;
149             }
150
151             /*
152              * Retreives the Itinerary which has been specifed by the user.
153              */
154             public Itinerary getItinerary() {
155                     Itinerary itinerary = new Itinerary();
156                     Enumeration enum = itsRows.elements();
157
158                     while (enum.hasMoreElements()) {
159                             Destination destination = (Destination)enum.nextElement();
160                             itinerary.addDestination(destination);
161                     }
162                     return itinerary;
163             }
164     }
```

```
1  /*      $Header: /com/meitca/hsl/zonesjagents/bootstrap/ItineraryPanel.java 3      11/11/96 5:13p Billp $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory,
4   *  Mitsubishi Electric Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  The ItineraryPanel of the AgentLaunchWizard
11  *
12  *      $Log: /com/meitca/hsl/zonesjagents/bootstrap/ItineraryPanel.java $
13  *
14  * 3     11/11/96 5:13p Billp
15  * Made correction to company name.
16  *
17  * 2     9/10/96 11:46a Walsh
18  * remove bogus imports that javac considers illegal
19  *
20  * 1     8/29/96 5:21p Walsh
21  * intial version
22  *
23  */
24 package com.meitca.hsl.zonesjagents.bootstrap;
25 import java.awt.*;
26 import java.io.*;
27 import com.roguewave.widgets.grid.*;
28
29 import com.meitca.hsl.util.Wizard;
30 import com.meitca.hsl.util.WizardPanel;
31
32
33 /**
34  * The third panel of the AgentLaunchWizard.  This panel allows the user
35  * to specify the Itinerary of the agent.
36  * @see         Itinerary
37  * @see         AgentLaunchWizardPanel
38  * @see         AgentLaunchWizard
39  * @author      Thomas Walsh
40  */
41 public class ItineraryPanel extends AgentLaunchWizardPanel {
42         /** The grid control on the panel */
43         Grid                                    itsGrid;
44
45         /** The data model object used in conjunction with itsGrid */
46         ItineraryGridDataModel  itsDataModel;
47
```

```
48          /**
49           * Construct the ItineraryPanel and all of its components.
50           * @param       wizard The wizard to which this panel belongs (needed by
51           *                      super)
52           * @param       wiinfozard The AgentLaunchInformation descringing this
53           *                      agent launch (needed by super)
54           */
55          public ItineraryPanel(Wizard wizard, AgentLaunchInformation info) {
56                  super(wizard, info);
57                  buildPanel();
58          }
59
60          /**
61           * The Wizard calls this method right before it is going to display
62           * the panel.  In this case, the Panel needs to recreate the Grid
63           * so that the ChoiceColumn of the grid contains the names of the
64           * available methods of the currently selected Agent.
65           */
66          public void displayPanel() {
67                  remove(itsGrid);
68
69                  // Crate a new grid model
70                  itsDataModel = new ItineraryGridDataModel(this);
71          // Crate a new Grid in spreadsheed mode.
72          itsGrid = new Grid(Grid.TABLE_MODE,itsDataModel,itsDataModel,itsDataModel,itsDataModel);
73          itsGrid.getCellPanel().setVertLineColor(Color.gray);
74          itsGrid.getCellPanel().setHorzLineColor(Color.gray);
75                  itsGrid.editable = true;
76
77                  // Set the grid column heading panel's attributes //
78          itsGrid.setColumnHeadingPanel(new ColumnHeadingPanel());
79          itsGrid.getColumnHeadingPanel().setBackground(Color.lightGray);
80          itsGrid.getColumnHeadingPanel().setForeground(Color.black);
81          itsGrid.getColumnHeadingPanel().setThreeDStyle(Grid.THREED_RAISED);
82          itsGrid.getColumnHeadingPanel().setHorzLines(true);
83          itsGrid.getColumnHeadingPanel().setTextLines(1);
84
85          // Add a text field column to the grid //
86          TextFieldColumnAttribute col1;
87          col1 = new TextFieldColumnAttribute();
88          col1.setTitle(StringResources.DEST_COL_TITLE);
89          col1.setWidth(150);
90          ((TableColumnCollection) itsGrid.getColumnCollection()).add(col1);
91
92          ChoiceColumnAttribute choiceColumn = new ChoiceColumnAttribute();
93          choiceColumn.setTitle(StringResources.ACTION_COL_TITLE);
94          choiceColumn.setWidth(150);
```

```
95          ((TableColumnCollection) itsGrid.getColumnCollection()).add(choiceColumn);
96
97
98          // Set the grid to have 1 row. //
99          (itsGrid.getRowCollection()).setRows(1);
100
101                 constrain(this, itsGrid,
102                                         0, 6, 1, 1,
103                                         GridBagConstraints.BOTH, GridBagConstraints.NORTHWEST, 1.0, 1.0,
104                                         0,0,0,0);
105
106
107                 choiceColumn.addItem("");
108                 for(int i=0; i<itsAgentInfo.itsMethods.length; i++) {
109                         choiceColumn.addItem(itsAgentInfo.itsMethods[i]);
110                 }
111
112                 invalidate();
113                 validate();
114         }
115
116         /**
117          * Verify that the information entered into the panel is valid.  Called
118          * after the "Next >" button is pressed.
119          * There is no actual verification necessary.  This method simply
120          * retireves the Itinerary from the data model and stuffs it into
121          * the AgentLaunchInfo.
122          * @returns     true if the panel contains valid info, false if not
123          */
124         public boolean validatePanel() {
125                 itsAgentInfo.itsItinerary = itsDataModel.getItinerary();
126                 return true;
127         }
128
129         /**
130          * Builds the GUI Components for this panel
131          */
132         private void buildPanel() {
133                 setLayout(new GridBagLayout());
134
135                 Label label1 = new Label(StringResources.ITIN_TEXT1, Label.LEFT);
136                 constrain(this, label1,
137                                         0, 0, 1, 1,
138                                         GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.
139                                         0,0,0,0);
140
141                 Label label2 = new Label(StringResources.ITIN_TEXT2, Label.LEFT);
```

```
142                 constrain(this, label2,
143                                   0, 1, 1, 1,
144                                   GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
145                                   0,0,0,0);
146
147
148                 Label label3 = new Label(StringResources.ITIN_TEXT3, Label.LEFT);
149                 constrain(this, label3,
150                                   0, 2, 1, 1,
151                                   GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
152                                   0,0,0,0);
153
154
155                 Label label4 = new Label(StringResources.ITIN_TEXT4, Label.LEFT);
156                 constrain(this, label4,
157                                   0, 3, 1, 1,
158                                   GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
159                                   0,0,0,0);
160
161                 Label label5 = new Label(StringResources.ITIN_TEXT5, Label.LEFT);
162                 constrain(this, label5,
163                                   0, 4, 1, 1,
164                                   GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
165                                   0,0,0,0);
166
167                 Label label6 = new Label(StringResources.ITIN_TEXT6, Label.LEFT);
168                 constrain(this, label6,
169                                   0, 5, 1, 1,
170                                   GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
171                                   0,0,0,0);
172
173                 // Crate a new grid model
174                 itsDataModel= new ItineraryGridDataModel(this);
175         // Crate a new Grid in spreadsheed mode.
176         itsGrid = new Grid(Grid.TABLE_MODE,itsDataModel,itsDataModel,itsDataModel,itsDataModel);
177         itsGrid.getCellPanel().setVertLineColor(Color.gray);
178         itsGrid.getCellPanel().setHorzLineColor(Color.gray);
179                 itsGrid.editable = true;
180
181                 // Set the grid column heading panel's attributes //
182         itsGrid.setColumnHeadingPanel(new ColumnHeadingPanel());
183         itsGrid.getColumnHeadingPanel().setBackground(Color.lightGray);
184         itsGrid.getColumnHeadingPanel().setForeground(Color.black);
185         itsGrid.getColumnHeadingPanel().setThreeDStyle(Grid.THREED_RAISED);
186         itsGrid.getColumnHeadingPanel().setHorzLines(true);
187         itsGrid.getColumnHeadingPanel().setTextLines(1);
188
```

```
189          // Add a text field column to the grid //
190          TextFieldColumnAttribute col1;
191          col1 = new TextFieldColumnAttribute();
192          col1.setTitle(StringResources.DEST_COL_TITLE);
193          col1.setWidth(150);
194          ((TableColumnCollection) itsGrid.getColumnCollection()).add(col1);
195
196          ChoiceCclumnAttribute choiceColumn = new ChoiceColumnAttribute();
197          choiceCclumn.setTitle(StringResources.ACTION_COL_TITLE);
198          choiceColumn.setWidth(150);
199          ((TableColumnCollection) itsGrid.getColumnCollection()).add(choiceColumn);
200
201
202          // Set the grid to have 1 row. //
203          (itsGrid.getRowCollection()).setRows(1);
204
205                  constrain(this, itsGrid,
206                                          0, 6, 1, 1,
207                                          GridBagConstraints.BOTH, GridBagConstraints.NORTHWEST, 1.0, 1.0,
208                                          0,0,0,0);
209
210
211          }
212
213          /**
214           * called by the ItineraryGridDataModel to signify that a new row should be
215           * added to the Grid
216           */
217          protected void addRow() {
218                  itsGrid.getRowCollection().setRows(itsGrid.getRowCollection().count()+1);
219          }
220  }
```

```
189         // Add a text field column to the grid //
190         TextFieldColumnAttribute col1;
191         col1 = new TextFieldColumnAttribute();
192         col1.setTitle(StringResources.DEST_COL_TITLE);
193         col1.setWidth(150);
194         ((TableColumnCollection) itsGrid.getColumnCollection()).add(col1);
195
196         ChoiceCclumnAttribute choiceColumn = new ChoiceColumnAttribute();
197         choiceCclumn.setTitle(StringResources.ACTION_COL_TITLE);
198         choiceColumn.setWidth(150);
199         ((TableColumnCollection) itsGrid.getColumnCollection()).add(choiceColumn);
200
201
202         // Set the grid to have 1 row. //
203         (itsGrid.getRowCollection()).setRows(1);
204
205                 constrain(this, itsGrid,
206                                         0, 6, 1, 1,
207                                         GridBagConstraints.BOTH, GridBagConstraints.NORTHWEST, 1.0, 1.0,
208                                         0,0,0,0);
209
210
211         }
212
213         /**
214          * called by the ItineraryGridDataModel to signify that a new row should be
215          * added to the Grid
216          */
217         protected void addRow() {
218                 itsGrid.getRowCollection().setRows(itsGrid.getRowCollection().count()+1);
219         }
220 }
```

```
 1  /*      $Header: /com/meitca/hsl/zonesjagents/bootstrap/NoAgentSkeletonException.java 2     11/11/96 5:13p Billp $
 2   *
 3   *  Copyright  1996 Horizon Systems Laboratory,
 4   *  Mitsubishi Electric Information Technology Center America.
 5   *      All rights reserved.
 6   *
 7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 8   *
 9   *  DESCRIPTION
10   *  Exception indicating that Agent invocation Skeleton is missing.
11   *
12   *      $Log: /com/meitca/hsl/zonesjagents/bootstrap/NoAgentSkeletonException.java $
13   *
14   * 2     11/11/96 5:13p Billp
15   * Made correction to company name.
16   *
17   * 1     8/29/96 5:21p Walsh
18   * intial version
19   *
20   */
21  package com.meitca.hsl.zonesjagents.bootstrap;
22
23  /**
24   * Exception indicates that the system could not locate the Agent
25   * method invocation skeleton.  This skeleton is used by the conduit
26   * server to invoke the proper method on the Agent when it arrives
27   * at a destination.  The skeleton is generated by the Agent Skeleton
28   * Generator.
29   * @see          Agent
30   * @see          AgentSkeleton
31   * @author       Thomas Walsh
32   */
33  public class NoAgentSkeletonException extends Exception {
34      public NoAgentSkeletonException() {
35                  super();
36      }
37
38      public NoAgentSkeletonException(String s) {
39                  super(s);
40      }
41  }
```

```
 1  /*      $Header: /com/meitca/hsl/zonesjagents/bootstrap/NotAgentException.java 3      11/11/96 5:13p Billp $
 2   *
 3   *  Copyright  1996 Horizon Systems Laboratory,
 4   *  Mitsubishi Electric Information Technology Center America.
 5   *      All rights reserved.
 6   *
 7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 8   *
 9   *  DESCRIPTION
10   *  Exception indicating that a .class File does not contain an Agent.
11   *
12   *      $Log: /com/meitca/hsl/zonesjagents/bootstrap/NotAgentException.java $
13   *
14   * 3     11/11/96 5:13p Billp
15   * Made correction to company name.
16   *
17   * 2     9/30/96 6:35p Walsh
18   * fix up javadoc comments
19   *
20   * 1     8/29/96 5:21p Walsh
21   * intial version
22   *
23   */
24  package com.meitca.hsl.zonesjagents.bootstrap;
25
26  /**
27   * Exception indicates the class File specified did not contain
28   * a Java class derived from Agent.
29   * @see         Agent
30   * @author      Thomas Walsh
31   */
32  public class NotAgentException extends Exception {
33      public NotAgentException() {
34                  super();
35      }
36
37      public NotAgentException(String s) {
38                  super(s);
39      }
40  }
```

WO 98/21662 PCT/US97/20232

```
1  /*      $Header: /com/meitca/hsl/zonesjagents/bootstrap/RelatedFilesPanel.java 3      11/11/96 5:13p Billp $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory,
4   *  Mitsubishi Electric Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  The RelatedFilesPanel of the AgentLaunchWizard
11  *
12  *      $Log: /com/meitca/hsl/zonesjagents/bootstrap/RelatedFilesPanel.java $
13  *
14  * 3     11/11/96 5:13p Billp
15  * Made correction to company name.
16  *
17  * 2     9/10/96 11:46a Walsh
18  * remove bogus imports that javac considers illegal
19  *
20  * 1     8/29/96 5:21p Walsh
21  * intial version
22  *
23  */
24 package com.meitca.hsl.zonesjagents.bootstrap;
25
26 import java.awt.*;
27 import java.io.*;
28
29 import com.meitca.hsl.util.Wizard;
30 import com.meitca.hsl.util.WizardPanel;
31
32
33 /**
34  * The second panel of the AgentLaunchWizard.  This panel allows the user
35  * to specify the filenames of classes that should be sent with the agent.
36  * @see         Agent
37  * @see         AgentLaunchWizardPanel
38  * @see         AgentLaunchWizard
39  * @author      Thomas Walsh
40  */
41 public class RelatedFilesPanel extends AgentLaunchWizardPanel {
42         /** The list control form which the user selects files */
43         List itsRelatedClassesList;
44
45
46         /**
47          * Construct the FindFilePanel and all of its components.
```

WO 98/21662 PCT/US97/20232

```
48          * @param       wizard The wizard to which this panel belongs (needed by
49          *                      super)
50          * @param       wiinfozard The AgentLaunchInformation descringing this
51          *                      agent launch (needed by super)
52          */
53         public RelatedFilesPanel(Wizard wizard, AgentLaunchInformation info) {
54                 super(wizard, info);
55                 buildPanel();
56         }
57
58         /**
59          * The Wizard calls this method right before it is going to display
60          * the panel.  In this case, the Panel needs to fill the List
61          * with the names of all the .class files in the same directory as
62          * the Agent.
63          */
64         public void displayPanel() {
65                 File agentDir = new File(itsAgentInfo.itsAgentDirectory);
66                 String[] files = agentDir.list(new ClassFileFilter());
67
68                 itsRelatedClassesList.clear();
69                 for (int i=0; i<files.length; i++) {
70                         if (!files[i].equals(itsAgentInfo.itsAgentFile))
71                                 itsRelatedClassesList.addItem(files[i]);
72                 }
73
74         }
75
76         /**
77          * Verify that the information entered into the panel is valid.  Called
78          * after the "Next >" button is pressed.
79          * There is no actual verification necessary.  This method simply
80          * retireves the list of classes and stuffs them into the
81          * AgentLaunchInfo
82          * @returns     true if the panel contains valid info, false if not
83          */
84         public boolean validatePanel() {
85                 itsAgentInfo.itsRelatedClasses = itsRelatedClassesList.getSelectedItems();
86                 return true;
87         }
88
89         /**
90          * Builds the GUI Components for this panel
91          */
92         private void buildPanel() {
93                 setLayout(new GridBagLayout());
94
```

WO 98/21662 PCT/US97/20232

```
95                  Label label1 = new Label(StringResources.RF_TEXT1, Label.LEFT);
96                  constrain(this, label1,
97                                          0, 0, 1, 1,
98                                          GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
99                                          0,0,0,0);
100
101                 Label label2 = new Label(StringResources.RF_TEXT2, Label.LEFT);
102                 constrain(this, label2,
103                                         0, 1, 1, 1,
104                                         GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
105                                         0,0,0,0);
106
107
108                 Label label3 = new Label(StringResources.RF_TEXT3, Label.LEFT);
109                 constrain(this, label3,
110                                         0, 2, 1, 1,
111                                         GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
112                                         0,0,0,0);
113
114
115                 Label label4 = new Label(StringResources.RF_TEXT4, Label.LEFT);
116                 constrain(this, label4,
117                                         0, 3, 1, 1,
118                                         GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
119                                         0,0,0,0);
120
121                 Label label5 = new Label(StringResources.RF_TEXT5, Label.LEFT);
122                 constrain(this, label5,
123                                         0, 4, 1, 1,
124                                         GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
125                                         0,0,0,0);
126
127                 Label label6 = new Label(StringResources.RF_TEXT6, Label.LEFT);
128                 constrain(this, label6,
129                                         0, 5, 1, 1,
130                                         GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
131                                         0,0,0,0);
132
133                 itsRelatedClassesList = new List(7, true);
134                 constrain(this, itsRelatedClassesList,
135                                         0, 6, 1, GridBagConstraints.REMAINDER,
136                                         GridBagConstraints.BOTH, GridBagConstraints.NORTHWEST, 1.0, 1.0,
137                                         0,0,0,0);
138         }
139 }
140
141 /**
```

WO 98/21662 PCT/US97/20232

```
142   * A FilenameFilter object than scans for file ending in .class
143   */
144  class ClassFileFilter implements FilenameFilter {
145          public boolean accept(File  dir, String  name) {
146                  return (name.endsWith(".class"));
147          }
148  }
```

```
/*      $Header: /com/meitca/hsl/zonesjagents/bootstrap/StringResources.java 7      11/11/96 5:13p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  The String Resources for the bootstap package
 *
 *      $Log: /com/meitca/hsl/zonesjagents/bootstrap/StringResources.java $
 *
 * 7     11/11/96 5:13p Billp
 * Made correction to company name.
 *
 * 6     10/07/96 4:44p Walsh
 * Save name & location of last agent launched.  Restart Wizard pointing
 * at same agent.
 *
 * 5     9/15/96 5:11p Walsh
 * Fix spelling mistakes
 *
 * 4     9/09/96 5:59p Walsh
 * Make constants final
 *
 * 3     9/05/96 6:37p Walsh
 * Add some string constants associated with the command line launch
 * utility.
 *
 * 2     9/04/96 12:36p Walsh
 * Add launch error message
 *
 * 1     8/29/96 5:21p Walsh
 * intial version
 *
 */
package com.meitca.hsl.zonesjagents.bootstrap;

class StringResources {

        // AgentLaunchWizard.java
        public static String WIZARD_TITLE = "Agent Launch Wizard";
        public static String WIZARD_APPNAME = "AgentLaunchWizard";
        public static String PROP_DESC = "Properties file for Zones Agent Launch Wizard";

        // FindFilePanel.java
```

WO 98/21662 PCT/US97/20232

```
    48          public static final String LAUNCH_ERROR_TITLE = "Launch Error";
    49          public static final String NO_FILENAME = "You must specify the filename of the .class file containing the a
nt.";
    50          public static final String ENDSWITH_CLASS = "You must specify a file ending with .class .";
    51          public static final String FF_TEXT1 = "This wizard will assist you in launching Agents";
    52          public static final String FF_TEXT2 = "Please select the .class file containing the ";
    53          public static final String FF_TEXT3 = "agent you would like to launch.";
    54          public static final String FF_TEXT4 = "                                                ";
    55          public static final String FF_FIELD_LABEL = "Class File: ";
    56          public static final String FF_BROWSE_BUTTON = "Browse...";
    57          public static final String BROWSE_DLG_TITLE = "Open Agent Class File";
    58
    59          //ItineraryPanel.java
    60          public static final String DEST_COL_TITLE = "Destination";
    61          public static final String ACTION_COL_TITLE = "Action";
    62          public static final String ITIN_TEXT1 = "An agent's itinerary specifies where an agent travels to and";
    63          public static final String ITIN_TEXT2 = "what operations the agent performs as it travels.  Please  ";
    64          public static final String ITIN_TEXT3 = "specify the itinerary below.  You should enter the host- ";
    65          public static final String ITIN_TEXT4 = "names of the machines you would like your agent to travel  ";
    66          public static final String ITIN_TEXT5 = "to as well as the methods you would like called at each stop.";
    67          public static final String ITIN_TEXT6 = "                                                        ";
    68
    69          // RelatedFilesPanel.java
    70          public static final String RF_TEXT1 = "You may specify a list of classes which should be sent with";
    71          public static final String RF_TEXT2 = "the Agent.  These should be classes which the Agent is likely";
    72          public static final String RF_TEXT3 = "to use while it is travelling.    ";
    73          public static final String RF_TEXT4 = "                                                  ";
    74          public static final String RF_TEXT5 = "Please select any classes you would like sent with the agent.";
    75          public static final String RF_TEXT6 = "                                                  ";
    76
    77          // BootStrap.java
    78          public static final String USAGE = "Usage: bootstrap [-d hostname,method] [-f relatedfile.class]\n\t[-h] ag
tfile.class";
    79          public static final String ILLEGAL_OPTN = "bootstrap: illegal option -- ";
    80          public static final String SYNTAX_ERROR = "bootstrap: syntax error -- ";
    81          public static final String ILLEGAL_AGENT = "bootstrap: Illegal agent file -- ";
    82          public static final String NO_AGENT = "bootstrap: No agent specified";
    83          public static final String AGENT_HELP = "\tThe agent files must be valid Java class file\n\tand must end wi
.class";
    84          public static final String ILLEGAL_RELFILE = "bootstrap: Illegal related filename -- ";
    85          public static final String RELFILE_HELP = "\tThe related files specified by the -f option\n\tmust be valid
va class files and must end\n\twith .class";
    86          public static final String ILLEGAL_DEST = "bootstrap: Illegal destination specification -- ";
    87          public static final String ILLEGAL_METHOD = "bootstrap: Illegal method name -- ";
    88          public static final String NO_DEST = "bootstrap: You must specify at least destination";
    89
    90          // shared strings
```

WO 98/21662 PCT/US97/20232

```
    91          public static final String FILE_NOTFOUND = "Could not find the file specified.";
    92          public static final String IOERROR = "An error occurred while trying to access the file specified.
    93          public static final String BAD_CLASS_FORMAT = "The file specified did not contain a valid Java cla
    94          public static final String NOT_AGENT = "The file specified did not contain a Java Agent.";
    95          public static final String NO_SKELETON = "Could not locate the Agent's skeleton.";
    96          public static final String LAUNCH_ERROR = "An error occurred while attempting to launch the Agent"
    97
    98  }
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/conduit/AgentPackage.java 18    11/11/96 5:15p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Agent transportation package object
 *
 *      $Log: /com/meitca/hsl/zonesjagents/conduit/AgentPackage.java $
 *
 * 18    11/11/96 5:15p Billp
 * Made correction to company name.
 *
 * 17    10/28/96 6:31p Walsh
 * Add more debugging info
 *
 * 16    10/24/96 5:15p Walsh
 * add some debugging information related to memory usage.
 *
 * 15    10/23/96 2:22p Walsh
 * add more Debug.println's
 *
 * 14    10/11/96 9:26a Walsh
 * fix compiler bug
 *
 * 13    10/10/96 6:02p Walsh
 * Add persistence support
 *
 * 12    9/30/96 7:06p Walsh
 * clean up javadoc comments
 *
 * 11    9/29/96 4:08p Walsh
 * Allow Destination methods to be specified by name
 * Give Agent access to AgentPackage information
 *
 * 10    9/11/96 12:24p Walsh
 * remove uneeded import of CollaboratorAgent
 *
 * 9     9/09/96 4:59p Walsh
 * Call Agent prepareForTransport, completedTransport methods
 *
 * 8     8/23/96 3:58p Walsh
 * Add support for ad-hoc method invocation
 *
```

```
48   * 7     8/22/96 5:45p Walsh
49   * Move the custom marshalling of the agent into the AgentPackage class
50   *
51   * 6     8/19/96 5:26p Walsh
52   * Add support for remote class loading
53   *
54   * 5     8/09/96 4:38p Walsh
55   * Setting up file headers
56   */
57
58  package com.meitca.hsl.zonesjagents.conduit;
59
60  import java.io.*;
61  import java.net.*;
62  import java.rmi.server.MarshalException;
63
64
65  import com.meitca.hsl.zonesjagents.shared.Agent;
66  import com.meitca.hsl.util.*;
67
68
69  /**
70   * An AgentPackage contains an Agent, its MobileCodebase and
71   * its Itinerary.  The AgentPackage is passes between ConduitServers
72   * when the Agent travels.
73   * @see         ConduitServerImpl
74   * @author      Thomas Walsh
75   */
76  public class AgentPackage {
77          Agent                                   itsAgent;
78          MobileCodebase                  itsMobileCodebase;
79      Itinerary                           itsItinerary;
80          String                                  itsHomeCodebaseURL;
81          int                                             itsOID;
82
83
84
85          /**
86           * Constructs an AgentPackage
87           * @param       agent The Agent
88           * @param       code The Agent's mobile codebase
89           * @param       itinerary The Agents itinerary
90           * @param       homeCodebaseURL The URL of the Agent's codebase
91           */
92          public AgentPackage(Agent agent, MobileCodebase codebase,
93                                                  Itinerary itinerary, String homeCodebaseURL) {
94                  Debug.println("conduit.memory", 3, "AgentPackage " + this + " created");
```

```
95              itsAgent = agent;
96              itsMobileCodebase = codebase;
97              itsItinerary = itinerary;
98                  itsHomeCodebaseURL = homeCodebaseURL;
99                  itsOID = 0;
100         }
101
102         protected void finalize() throws Throwable {
103                 Debug.println("conduit.memory", 3, "AgentPackage " + this + " finalized");
104                 super.finalize();
105         }
106
107         /** Retrieves the Agent from the package */
108         public Agent getAgent() {
109             return itsAgent;
110         }
111
112         /** Retrieves the Agent's Mobile Codebase from the package */
113         public MobileCodebase getMobileCodebase() {
114             return itsMobileCodebase;
115         }
116
117         /** Retrieves the Agent's itinerary from the package */
118         public Itinerary getItinerary() {
119             return itsItinerary;
120         }
121
122         /** Retrieves the Agent's codebase URL  from the package */
123         public String getHomeCodebaseURL() {
124             return itsHomeCodebaseURL;
125         }
126
127         /**
128          * Prepares the Agent package for network transmission.  This method
129          * performs the conversion of the Agent to a network transmission
130          * format.  This method also calls the Agent's prepareForTransport
131          * method.
132          */
133         public void prepareForTransport() {
134                 Debug.println("conduit", 2, "AgentPackage.prepareForTransport called");
135
136                 // give the agent a chance to do any work in preperation
137                 // for transport
138                 itsAgent.prepareForTransport();
139         }
140
141         /**
```

WO 98/21662 PCT/US97/20232

```
142          * Restores the Agent package for network transmissionfrom its
143          * network transmission format.  This method also calls the Agent's
144          * completedTransport method.
145          */
146         public void restoreFromTransportForm() {
147                 Debug.println("conduit", 2, "AgentPackage.restoreFromTransportForm called");
148
149                 // give the agent a chance to do any work in preperation
150                 // for arrival at destination
151                 itsAgent.completedTransport();
152         }
153
154         private void readObject(ObjectInputStream stream)
155                                         throws IOException, ClassNotFoundException {
156                 Debug.println("conduit", 2, "    AgentPackage.readObject called");
157
158                 itsMobileCodebase = (MobileCodebase)stream.readObject();
159                 Debug.println("conduit", 4, "    read MobileCodebase: " + itsMobileCodebase);
160
161                 itsItinerary = (Itinerary)stream.readObject();
162                 Debug.println("conduit", 4, "    read Itinerary: " + itsItinerary);
163
164                 itsHomeCodebaseURL = (String)stream.readObject();
165                 Debug.println("conduit", 4, "    read Codebase URL: " + itsHomeCodebaseURL);
166
167                 // do the special deserialization of the Agent using the
168                 // ConduitObjectInputStream class
169                 ConduitServerClassLoader classLoader = new ConduitServerClassLoader(itsMobileCodebase, itsHomeCodeb
eURL);
170
171                 // unmarshal the agent, build the AgentPackage and then pass it along to receiveAgent
172                 byte[]                                      serializedAgent = (byte[])stream.readObject();
173                 ByteArrayInputStream            byteInStream = new ByteArrayInputStream(serializedAgent);
174                 ConduitObjectInputStream        marshalInput = new ConduitObjectInputStream(byteInStream, classLoad
);
175
176                 Debug.println("conduit", 4, "    about to read Agent");
177                 itsAgent = (Agent)marshalInput.readObject();
178                 Debug.println("conduit", 4, "    read Agent: " + itsAgent);
179
180                 // Restore the Agents reference to its Package will be
181                 // restored when the agent arrives at the Destination
182                 itsAgent.setPackage(this);
183         }
184
185         private void writeObject(ObjectOutputStream stream)
186                                         throws IOException {
```

WO 98/21662 PCT/US97/20232

```
187                     Debug.println("conduit", 2, "AgentPackage.writeObject called");
188
189                     stream.writeObject(itsMobileCodebase);
190                     stream.writeObject(itsItinerary);
191                     stream.writeObject(itsHomeCodebaseURL);
192
193                     // do the special deserialization of the Agent using the
194                     // ConduitObjectOutputStream class
195
196                     // null out the Agents pointer to the AgentPackage.  We have to
197                     // do this because Java object persistence does not support
198                     // isomorphic persistence.  Since the AgentPackage has a
199                     // reference to the Agent and the Agent has a reference
200                     // to the AgentPackage we have a circular reference.
201                     // Without isomorphic persistence, this circular referece
202                     // would cause the object serialization code to go into an
203                     // infinite loop.  The reference to itsPackage will be
204                     // restored when the agent arrives at the Destination
205                     itsAgent.setPackage(null);
206
207                     ByteArrayOutputStream           byteOutStream = new ByteArrayOutputStream();
208                     ConduitObjectOutputStream       marshalOutput = new ConduitObjectOutputStream(byteOutStream);
209
210                     // use object serialization to stream the agent to a byte array
211                     marshalOutput.writeObject(itsAgent);
212                     byte[]  serializedAgent = byteOutStream.toByteArray();
213
214                     // finally write the serialized agent to the original output stream
215                     stream.writeObject(serializedAgent);
216             }
217
218             /** Retrieves the AgentSkeleton object corresponding to the Agent. */
219             AgentSkeleton getAgentSkeleton() throws ClassNotFoundException, InstantiationException,  IllegalAccessExcep
220             {
221                     String agentClassname = itsAgent.getClass().getName();
222                     String skelClassname = agentClassname + "_Skel";
                        ConduitServerClassLoader classLoader = (ConduitServerClassLoader)itsAgent.getClass().getClassLoader
223                     if (classLoader != null)
224                             return (AgentSkeleton)(classLoader.loadClass(skelClassname).newInstance());
225                     else
226                             return (AgentSkeleton)(Class.forName(skelClassname).newInstance());
227             }
228
229             void setOID(int oid) {
230                     itsOID = oid;
231             }
```

WO 98/21662 PCT/US97/20232

AgentPackage.java_1 Tue Nov 12 07:45:01 1996 6

```
232
233         int getOID() {
234             return itsOID;
235         }
236     }
```

```
 1  /*  $Header: /com/meitca/hsl/zonesjagents/conduit/AgentSkeleton.java 3     11/11/96 5:15p Billp $
 2   *
 3   *  Copyright  1996 Horizon Systems Laboratory,
 4   *  Mitsubishi Electric Information Technology Center America.
 5   *      All rights reserved.
 6   *
 7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 8   *
 9   *  DESCRIPTION
10   *  Agent method invocation skeleton interface
11   *
12   *      $Log: /com/meitca/hsl/zonesjagents/conduit/AgentSkeleton.java $
13   *
14   * 3     11/11/96 5:15p Billp
15   * Made correction to company name.
16   *
17   * 2     8/29/96 5:15p Walsh
18   * Add getMethods method
19   *
20   * 1     8/23/96 3:50p Walsh
21   * Initial versions
22   */
23  package com.meitca.hsl.zonesjagents.conduit;
24
25  import com.meitca.hsl.zonesjagents.shared.Agent;
26
27  /**
28   * An interface representing an Agent ad-hoc method invocation skeleton.
29   * The skeleton allows the ConduitServer to make a call to any method within
30   * the Agent.  At runtime, the ConduitServer looks at the Agent's Itinerary
31   * and determines what method should be called at the current location.  The
32   * skeleton allows the Conduit Server to call the method.
33   * @see          Agent
34   * @see          ConduitServer
35   * @author       Thomas Walsh
36   */
37  public interface AgentSkeleton {
38
39          /**
40           * Invokes a method on an agent
41           * @param       agent The agent whose method to invoke.
42           * @param       i       The method ID number of the method to invoke.
43           */
44          public void invoke(Agent agent, int i) throws IllegalAccessException;
45
46          /**
47           * Retrieves a list of the agent methods which can be invoked by
```

WO 98/21662 PCT/US97/20232

AgentSkeleton.java_1 Tue Nov 12 07:45:01 1996 2

```
48          * the skeleton.
49          */
50         public String[] getMethods();
51
52     }
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/conduit/AgentThread.java 4     11/11/96 5:15p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  provides the thread in which an Agent will executes.
 *
 *      $Log: /com/meitca/hsl/zonesjagents/conduit/AgentThread.java $
 *
 * 4     11/11/96 5:15p Billp
 * Made correction to company name.
 *
 * 3     10/29/96 5:28p Walsh
 * Use ErrorLog class to report errors
 *
 * 2     10/28/96 6:31p Walsh
 * Add more debugging info
 *
 * 1     10/25/96 5:02p Walsh
 * Make AgentThread public
 *
 */
package com.meitca.hsl.zonesjagents.conduit;

import java.util.*;

import com.meitca.hsl.zonesjagents.shared.*;
import com.meitca.hsl.util.*;

/**
 * A subclass of Thread.  AgentThread provides the thread in which an Agent will
 * execute.
 * @see         Agent
 * @see         ConduitServer
 * @author      Thomas Walsh
 */
public class AgentThread extends Thread {
        AgentPackage        itsPackage;

        /**
     * Constructs an AgentThread
     * @param           agentPackage The package containing the agent, its code and
       *                                  its itinerary.
```

```
48        */
49           public AgentThread(AgentPackage agentPackage) {
50                   Debug.println("conduit.memory", 3, "AgentThread " + this + " created");
51               itsPackage =  agentPackage;
52           this.start();
53        }
54
55           protected void finalize() throws Throwable {
56                   Debug.println("conduit.memory", 3, "AgentThread " + this + " finalized");
57                   super.finalize();
58           }
59
60           /**
61            * Retrieves a reference to the Agent which is executing within this
62            * AgentThread
63            */
64           public Agent getAgent() {
65                   return  itsPackage.getAgent();
66           }
67
68           /**
69            * Retrieves a reference to the AgentPackage contained within this
70            * AgentThread
71            */
72           public AgentPackage getAgentPackage() {
73                   return  itsPackage;
74           }
75
76           /**
77         * Begins the Agent's execution
78         */
79           public void run() {
80               try {
81                           ConduitServerImpl.getPersistenceStore().addToStore(itsPackage);
82                           AgentSkeleton skel = itsPackage.getAgentSkeleton();
83                           Destination current = itsPackage.getItinerary().getCurrentLocation();
84
85                           if (current.getMethodID() == Destination.UNKNOWN_METHOD_ID) {
86                                   current.convertNameToId(skel);
87                           }
88
89                           skel.invoke(itsPackage.getAgent(),
90                                   current.getMethodID());
91                           itsPackage.getItinerary().getCurrentLocation().setCompleted();
92                           ConduitServerImpl.getPersistenceStore().updateStore(itsPackage);
93                   } catch (Exception error) {
94                           ErrorLog.println("conduit.AgentThread",
```

```
                                        ErrorLog.SEVERITY_ERROR,
                                        "An error occurred while executing Agent:",
                                        error);
                }

                ConduitServerImpl.decrementAgentCounters();

                try {
                        // pass the agent along.
                        ConduitServerImpl.sendPackage(itsPackage);
                } catch (NoSuchElementException done) {
            Debug.println("conduit", 1, "AgentThread.run: No more destinations for Agent " +
                                                                        itsPackage.getAgent().get
tID());
                } catch (Exception error) {
                        ErrorLog.println("conduit.AgentThread",
                                        ErrorLog.SEVERITY_ERROR,
                                        "An error occurred while transporting Agent:",
                                        error);
            }

                // remove the agent from the persistence store
                ConduitServerImpl.getPersistenceStore().removeFromStore(itsPackage);
        }
    }
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/conduit/ConduitObjectInputStream.java 9     11/11/96 5:15p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Specialized ObjectInputStream used for agent deserialization
 *
 *      $Log: /com/meitca/hsl/zonesjagents/conduit/ConduitObjectInputStream.java $
 *
 * 9     11/11/96 5:15p Billp
 * Made correction to company name.
 *
 * 8     11/01/96 5:32p Walsh
 * Modify debugging levels of debug output
 *
 * 7     10/21/96 4:26p Walsh
 * Add debug logging
 *
 * 6     9/30/96 7:06p Walsh
 * clean up javadoc comments
 *
 * 5     9/11/96 6:40p Walsh
 * fix compiler error
 *
 * 4     9/11/96 6:33p Walsh
 * derive from Marshal*Stream
 *
 * 3     8/22/96 2:19p Walsh
 * Clean up code a little
 *
 * 2     8/13/96 5:09p Walsh
 * Added comments & documentation.  resolveClass now uses
 * ConduitServerClassLoader
 *
 * 1     8/12/96 7:17p Walsh
 * Initial versions
 */
package com.meitca.hsl.zonesjagents.conduit;

import java.io.InputStream;
import java.io.IOException;
import java.io.StreamCorruptedException;
import java.lang.ClassNotFoundException;
```

```
 1  /*  $Header: /com/meitca/hsl/zonesjagents/conduit/ConduitObjectOutputStream.java 7     11/11/96 5:15p Billp $
 2   *
 3   *  Copyright  1996 Horizon Systems Laboratory,
 4   *  Mitsubishi Electric Information Technology Center America.
 5   *      All rights reserved.
 6   *
 7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 8   *
 9   *  DESCRIPTION
10   *  Specialized MershalInputStream used for agent deserialization
11   *
12   *      $Log: /com/meitca/hsl/zonesjagents/conduit/ConduitObjectOutputStream.java $
13   *
14   * 7     11/11/96 5:15p Billp
15   * Made correction to company name.
16   *
17   * 6     11/01/96 5:32p Walsh
18   * Modify debugging levels of debug output
19   *
20   * 5     10/21/96 4:26p Walsh
21   * Add debug logging
22   *
23   * 4     9/30/96 7:06p Walsh
24   * clean up javadoc comments
25   *
26   * 3     9/11/96 6:33p Walsh
27   * derive from Marshal*Stream
28   *
29   * 2     8/22/96 2:19p Walsh
30   * Clean up code a little
31   *
32   * 1     8/12/96 7:17p Walsh
33   * Initial versions
34   */
35  package com.meitca.hsl.zonesjagents.conduit;
36
37  import java.io.OutputStream;
38  import java.io.IOException;
39  import java.io.StreamCorruptedException;
40  import java.lang.ClassNotFoundException;
41  import java.rmi.server.MarshalOutputStream;
42
43  import com.meitca.hsl.util.*;
44
45  public class ConduitObjectOutputStream
46          extends  MarshalOutputStream {
47
```

WO 98/21662 PCT/US97/20232

```
ConduitObjectInputStream.java_1      Tue Nov 12 07:45:02 1996      3

    95          Debug.println("conduit.classload", 3, "ConduitObjectInputStream.resolveClass called for " + classr.
    );
    96
    97          return itsClassLoader.loadClass(classname);
    98      }
    99  }
```

```
48  import java.rmi.server.MarshalInputStream;
49  import java.net.MalformedURLException;
50  import java.net.URL;
51  import java.rmi.server.StubClassLoader;
52
53  import com.meitca.hsl.util.*;
54
55  /**
56   * A subclass of ObjectInputStream which is used to handle deserialization
57   * of Agent objects.  ConduitObjectInputStream overrides the
58   * ObjectInputStream.resolveClass method.  As the deserialization process
59   * encounters object contained within the agent, resolveClass is called for
60   * each unique class.  resolveClass makes use of a ConduitServerClassLoader
61   * object to search the agents MobileCodebase for class bytecodes.
62   * @see         Agent
63   * @see         ConduitServerClassLoader
64   * @see         ObjectInputStream
65   * @see         MobileCodebase
66   * @author      Thomas Walsh
67   */
68  class ConduitObjectInputStream
69          extends  MarshalInputStream {
70
71          /** Class loader object used for deserializing an agent. */
72          ConduitServerClassLoader itsClassLoader;
73
74          /**
75           * Constructs a ConduitObjectInputStream
76           * @param       is The InputStream containing the Agent object.
77           * @param       classLoader A ConduitServerClassLoader capable of loading
78           *                      classes from the Agent's MobileCodebase
79           */
80          public ConduitObjectInputStream(InputStream is, ConduitServerClassLoader classLoader)
81                  throws IOException, StreamCorruptedException {
82              super(is);
83
84                  itsClassLoader = classLoader;
85           }
86
87          /**
88           * Resolves a fully qualified class name to a Class object.  resolveClass uses a
89           * ConduitServerClassLoader to actually perform the class loading.
90           * @param       classname The classes fully qualified class name.
91           */
92          protected Class resolveClass(String classname)
93                      throws IOException, ClassNotFoundException {
94
```

WO 98/21662 PCT/US97/20232

```
48          public ConduitObjectOutputStream(OutputStream is)
49                  throws IOException {
50              super(is);
51          }
52
53
54          protected vcid annotateClass(Class cl) throws IOException {
55                  Debug.println("conduit.classload", 3, "ConduitObjectOutputStream.annotateClass called for " + cl);
56          }
57  }
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/conduit/ConduitServer.java 9     11/11/96 5:15p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Conduit Server distributed interface
 *
 *      $Log: /com/meitca/hsl/zonesjagents/conduit/ConduitServer.java $
 *
 * 9     11/11/96 5:15p Billp
 * Made correction to company name.
 *
 * 8     9/30/96 7:06p Walsh
 * clean up javadoc comments
 *
 * 7     8/27/96 3:51p Walsh
 * remove getConduitName method from ConduitServer interface
 *
 * 6     8/22/96 5:45p Walsh
 * Move the custom marshalling of the agent into the AgentPackage class
 *
 * 5     8/19/96 5:26p Walsh
 * Add support for remote class loading
 *
 * 4     8/13/96 11:47a Walsh
 * Add documentation & comments.  Change signature of receivePackage
 * method.
 *
 * 3     8/12/96 7:15p.Walsh
 * ConduitServer now performs Marshalling
 *
 * 2     8/09/96 4:38p Walsh
 * Setting up file headers
 */

package com.meitca.hsl.zonesjagents.conduit;

import java.rmi.Remote;
import java.rmi.RemoteException;

import com.meitca.hsl.zonesjagents.conduit.AgentPackage;
import com.meitca.hsl.zonesjagents.conduit.MobileCodebase;

```

WO 98/21662 PCT/US97/20232

```
48
49  /**
50   * ConduitServer is an RMI distributed interface.  It provides some RMI glue
51   * that allows conduit servers to communicate.  ConduitServer is implemented
52   * by the ConduitServerImpl class.  Conduit servers provide for Agent travel
53   * and execution
54   * @see         ConduitServerImpl
55   * @author      Thomas Walsh
56   */
57  public interface ConduitServer extends Remote {
58
59      /**
60       * Receive an Agent from the previous conduit server in its Itinerary.
61       * @param           aPackage The package containing the agents, its mobile
62            *                              codebase and it itinerary.
63       * @exception       RemoteException If a communication error occurs
64       */
65      public void receivePackage(AgentPackage aPackage)
66                                                  throws RemoteException;
67  }
```

WO 98/21662 PCT/US97/20232

```
/*  $Header: /com/meitca/hsl/zonesjagents/conduit/ConduitServerClassLoader.java 12    11/11/96 5:15p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Conduit Server ClassLoader subclass
 *
 *      $Log: /com/meitca/hsl/zonesjagents/conduit/ConduitServerClassLoader.java $
 *
 * 12    11/11/96 5:15p Billp
 * Made correction to company name.
 *
 * 11    11/01/96 5:32p Walsh
 * Modify debugging levels of debug output
 *
 * 10    10/24/96 5:15p Walsh
 * add some debugging information related to memory usage.
 *
 * 9     10/21/96 4:26p Walsh
 * Add debug logging
 *
 * 8     9/30/96 7:06p Walsh
 * clean up javadoc comments
 *
 * 7     9/28/96 6:37p Walsh
 * use the AgentCodebase class to retrieve bytecodes
 *
 * 6     8/22/96 5:42p Walsh
 * Cache Classes we define
 *
 * 5     8/19/96 5:25p Walsh
 *
 * 4     8/13/96 5:09p Walsh
 * Added comments & documentation.  loadClass now looks in MobileCodebase.
 *
 * 3     8/13/96 11:19a Walsh
 * remove obsolete loadMobileClasses method
 *
 * 2     8/09/96 4:38p Walsh
 * Setting up file headers
 */

package com.meitca.hsl.zonesjagents.conduit;
```

WO 98/21662 PCT/US97/20232

```
48
49  import java.rmi.Naming;
50  import java.io.*;
51  import java.net.*;
52  import java.util.*;
53
54  import com.meitca.hsl.zonesjagents.shared.*;
55  import com.meitca.hsl.util.*;
56
57  /**
58   * A specialized ClassLoader object used by the conduit server to manage agent
59   * creation, deserialization, and sub-object creation.  ConduitServerClassLoader
60   * extends the base system class loading behavior by allowing for classes to be
61   * loaded from the MobileCodebase of an Agent.  For example, an Agent object might
62   * be created on one particular server on a network.  When it is created, its
63   * bytecodes are added to the MobileCodebase.  The agent now travels to a new
64   * server.  The bytecodes for this particular agent have not been installed on the
65   * new server.  Under normal circumstances, when the object deserialization code
66   * attmepts to reconstruct the agent, a ClassNotFoundException would result since
67   * the agents code does not exist anywhere on the new server.
68   * ConduitServerClassLoader allows the Java virtual machine to look for an Agents
69   * code in the MobileCodebase which allows Agent object to travel and pull their
70   * code with them.
71   * @see         Agent
72   * @see         ConduitServer
73   * @see         MobileCodebase
74   * @see         ClassLoader
75   * @author      Thomas Walsh
76   */
77  public class ConduitServerClassLoader
78                      extends ClassLoader {
79
80      /** The MobileCodebase corresponding to the Agent being managed. */
81          MobileCodebase itsMobileCodebase;
82
83      /** An URL back to the coebase of this agent on its home machine */
84          AgentCodebase itsHomeCodebase;
85
86          /**
87           * A hashtable containing the Class objects we have already
88           * defined during the serialization process
89           */
90          Hashtable itsPreviouslyDefinedClasses;
91
92          /**
93           * Constructs a ConduitServerClassLoader
94           * @param               codebase The MobileCodebase corresponding to the
```

```
95           *                                   Agent being managed.
96          * @param                homeCodebaseURL An URL back to the codebase of this
97           *                                   agent on its home machine.
98           * @exception   MalformedURLException if the URL passesd in as the
99           *                                   homeCodebaseURL parameter is invalid.
100          */
101         public ConduitServerClassLoader(MobileCodebase codebase, String homeCodebaseURL)
102                                                          throws MalformedURLException {
103                 Debug.println("conduit.memory", 3, "ConduitServerClassLoader " + this + " created");
104                 itsMobileCodebase = codebase;
105                 itsHomeCodebase = new AgentCodebase(homeCodebaseURL);
106                 itsPreviouslyDefinedClasses = new Hashtable();
107         }
108
109         protected void finalize() throws Throwable {
110                 Debug.println("conduit.memory", 3, "ConduitServerClassLoader " + this + " finalized");
111                 super.finalize();
112         }
113
114
115     /**
116          * Loads a class.  First check on the local system and then looks in the
117          * MobileCodebase.
118          * @param       name Fully qualified name of the class to be loaded (ie <b>
119          *                   "java.lang.String"</b>).
120          */
121         public Class loadClass(String name)
122             throws ClassNotFoundException {
123         return loadClass(name, true);
124     }
125
126     /**
127          * Loads a class.  Inherited from ClassLoader. First check on the local
128          * system and then looks in the MobileCodebase.
129          * @param       name Fully qualified name of the class to be loaded (ie
130          *                   java.lang.String.
131          * @param       resolve boolean value indicating wether or not the class
132          *                   should be resolved via the ClassLoader.resolveClass method
133          */
134         protected Class loadClass(String  name, boolean  resolve)
135                 throws ClassNotFoundException {
136
137                 Debug.println("conduit.classload", 2, "ConduitServerClassLoader.loadClass called for " + nam
138                 Debug.println("conduit.classload", 2, "resolve is " + resolve);
139
140                 // first check the cache to see if we have encountered this class
141                 // before.  Note: A ClassLoader object should always do something
```

```
142                 // like this.  In order for Java's type checking to work properly,
143                 // everytime a loadClass is called for a particular class name, the
144                 // EXACT SAME Class object must be returned.
145                 Class cache = (Class)itsPreviouslyDefinedClasses.get(name);
146                 if (cache != null) {
147                         Debug.println("conduit.classload", 3, "Class found in cache");
148                         return cache;
149                 }
150
151                 try {
152                         // first check this system
153                         Class c = findSystemClass(name);
154                         Debug.println("conduit.classload", 3, "Class found on system");
155                         if (resolve) {
156                                 resolveClass(c);
157                         }
158                         return c;
159                 } catch (ClassNotFoundException e) {
160                         // wasn't found on this system.  Check the MobileCodebase...
161                         byte[] bytecodes = itsMobileCodebase.getCode(name);
162
163                         if (bytecodes != null) {
164                                 Debug.println("conduit.classload", 3, "Class found in MobileCodebase");
165
166                                 // bytecodes for the class were found in the MobileCodebase.
167                                 // Try to define a class from them.
168                                 Class c = defineClass(bytecodes, 0, bytecodes.length);
169                                 if (resolve) {
170                                         resolveClass(c);
171
172                                         // store this class in the cache
173                                         itsPreviouslyDefinedClasses.put(name, c);
174                                 }
175                                 return c;
176                         } else {
177                                 Debug.println("conduit.classload", 3, "Looking in home codebase");
178                                 return loadClassFromHomeCodebase(name, resolve);
179                         }
180                 }
181         }
182
183
184         /**
185          * Loads a class using the remote class loader.
186          * @param       name Fully qualified name of the class to be loaded (ie
187          *                      java.lang.String.
188          * @param       resolve boolean value indicating wether or not the class
```

```
189             *                       should be resolved via the ClassLoader.resolveClass method
190             */
191           protected Class loadClassFromHomeCodebase(String  name, boolean  resolve)
192                         throws ClassNotFoundException {
193
194                   Debug.println("conduit.classload", 2, "ConduitServerClassLoader.loadClassFromHomeCodebase called fo
" + name);
195                   Debug.println("conduit.classload", 2, "resolve is " + resolve);
196
197                   byte[] bytecodes = itsHomeCodebase.retrieveCode(name);
198
199                   // bytecodes for the class were found in at remote location.
200                   // store them in the MobileCodebase for future reference
201                   itsMobileCodebase.storeCode(name, bytecodes);
202
203
204                   // Try to define a class from them.
205                   Class c = defineClass(bytecodes, 0, bytecodes.length);
206                   if (resolve) {
207                         resolveClass(c);
208
209                         // store this class in the cache
210                         itsPreviouslyDefinedClasses.put(name, c);
211                   }
212           return c;
213           }
214   }
```

WO 98/21662 PCT/US97/20232

```
/*  $Header: /com/meitca/hsl/zonesjagents/conduit/ConduitServerImpl.java 23    11/11/96 5:15p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Conduit Server implementation
 *
 *      $Log: /com/meitca/hsl/zonesjagents/conduit/ConduitServerImpl.java $
 *
 * 23    11/11/96 5:15p Billp
 * Made correction to company name.
 *
 * 22    10/30/96 2:52p Walsh
 * Allow Debug and ErrorLog settings to be set in properties file
 *
 * 21    10/29/96 5:28p Walsh
 * Use ErrorLog class to report errors
 *
 * 20    10/28/96 6:31p Walsh
 * Add more debugging info
 *
 * 19    10/25/96 5:02p Walsh
 * Make AgentThread public.  Move to own source file.
 *
 * 18    10/24/96 5:15p Walsh
 * add some debugging information related to memory usage.
 *
 * 17    10/23/96 2:22p Walsh
 * add more Debug.println's
 *
 * 16    10/21/96 4:26p Walsh
 * Add debug logging
 *
 * 15    10/11/96 2:20p Walsh
 * remove restarted agents from persistence store
 *
 * 14    10/10/96 6:02p Walsh
 * Add persistence support
 *
 * 13    9/29/96 4:08p Walsh
 * Allow Destination methods to be specified by name
 * Give Agent access to AgentPackage information
 *
```

WO 98/21662 PCT/US97/20232

```
48   * 12    9/09/96 6:02p Walsh
49   * make constant final
50   *
51   * 11    9/04/96 12:32p Walsh
52   * Centralize manipultation of ConduitServer's RMI URL.
53   * Move test security manager out of this file.
54   *
55   * 10    8/30/96 3:53p Walsh
56   * null out itsPackage (in AgentThread.run) to try to nudge the garbage
57   * collector
58   *
59   * 9     8/23/96 3:58p Walsh
60   * Add support for ad-hoc method invocation
61   *
62   * 8     8/22/96 6:36p Walsh
63   * Add static reference to the one ConduitServer that exists.
64   *
65   * 7     8/22/96 5:45p Walsh
66   * Move the custom marshalling of the agent into the AgentPackage class
67   *
68   * 6     8/19/96 5:26p Walsh
69   * Add support for remote class loading
70   *
71   * 5     8/13/96 5:12p Walsh
72   * Changed signature of receivePackage.  Changed marshalling,
73   * un-marshalling code to marhsal only Agent and not entire AgentPackage
74   *
75   * 4     8/13/96 10:35a Walsh
76   * Add some comments and documentation
77   *
78   * 3     8/12/96 7:16p Walsh
79   * ConduitServer now performs Marshalling
80   *
81   * 2     8/09/96 4:38p Walsh
82   * Setting up file headers
83   */
84  package com.meitca.hsl.zonesjagents.conduit;
85
86  import java.rmi.server.UnicastRemoteServer;
87  import java.rmi.RemoteException;
88  import java.rmi.Naming;
89  import java.rmi.NotBoundException;
90  import java.net.*;
91  import java.io.*;
92  import java.util.*;
93
94  import com.meitca.hsl.zonesjagents.shared.*;
```

WO 98/21662 PCT/US97/20232

```
 95  import com.meitca.hsl.util.*;
 96  import com.meitca.hsl.zonesjagents.security.InsecurityManager;
 97
 98  /**
 99   * An RMI distributed object which provides for Agent travel and execution.
100   * ConduitServerImpl implements the ConduitServer interface.  ConduitServer
101   * is an RMI distributed interface.  ConduitServerImpl is responsible for
102   * receiving incoming agents, providing for execution of agents, and for
103   * transmitting agents to other destinations.
104   * @see         Agent
105   * @see         ConduitServer
106   * @author      Thomas Walsh
107   */
108  public class ConduitServerImpl
109                      extends UnicastRemoteServer
110                          implements ConduitServer {
111
112          /**
113           * The conduit server is a Sentinel object.  There will only be one of
114           * them running within a given Java vertual machine.  The static member
115           * theServer is a pointer to that one ConduitServerImpl object
116           */
117          static ConduitServerImpl        theServer;
118
119      /** The conduit servers name. */
120      String                                                  itsName;
121
122          /** The conduit servers properties file */
123          JASProperties                                   itsProperties;
124
125          /** An object representing the servers persistence store */
126          ConduitServerPersistenceStore   itsPersistenceStore;
127
128          /** String constants. */
129          /**The name to which a conduit server is bound in the RMI Registry */
130          static final String                     RMI_NAME = "ConduitServer";
131          static final String                     PROP_DESC= "Properties File for Conduit Server";
132
133
134      /*
135           * The following member variables store some statistical information
136           * about the performance of the Conduit Server.
137           */
138          /** The date and time when this Conduit Server was started */
139          Date                                                    itsStartTime;
140
141          /** The number of Agents currently in the system */
```

```
142         long                                                    itsCurrentAgentCount;
143
144         /**
145          * The total number of Agents to travel through the system
146          * since startup
147          */
148         long                                                    itsTotalAgentCount;
149
150         /**
151          * The largest number of Agent to be in the system at any one time.
152          */
153         long                                                    itsPeakAgentCount;
154
155
156
157         /**
158          * Constructs a conduit server
159          * @param               name The servers name.
160          * @exception   RemoteException If an error occurs setting up network
161          *                              communications
162          */
163         public ConduitServerImpl(String name) throws RemoteException {
164             itsName = new String(name);
165                 theServer = this;
166
167                 // intialize the counters
168                 itsTotalAgentCount = 0;
169                 itsCurrentAgentCount = 0;
170                 itsPeakAgentCount = 0;
171
172                 try {
173                         itsProperties = new JASProperties(RMI_NAME,
174
175                                                                                 JASProperties.SERVER,
176
177                 ) catch (Exception e) {                                         PROP_DESC,
178                                                                                 false);
179                         ErrorLog.println("conduit.ConduitServerImpl", ErrorLog.SEVERITY_WARNING,
180                                                         "Could not access Properties file for Conduit Server
181                         ErrorLog.println("conduit.ConduitServerImpl", ErrorLog.SEVERITY_WARNING,
182                                                         "Continuing with defaults.", e);
183
184                         // Construct an empty Properties object.
185                         itsProperties = new JASProperties();
186                 }
187
188                 // Set up the debug and error logging facilities using the
                    // Conduit Server specific properties.
```

WO 98/21662 PCT/US97/20232

```
189                 Debug.enable(itsProperties);
190                 ErrorLog.initialize(itsProperties);
191
192                 itsPersistenceStore = new ConduitServerPersistenceStore();
193         }
194
195
196         /**
197          * Send the Agent on to the next conduit server in its Itinerary.
198          * @param           agentPackage The Agent Package containing the Agent.
199          * @exception       NoSuchElementException If the agent has completed its
200            *                                       Itinerary.
201          * @exception       RemoteException If an error occurs communicating with
202            *                                       the next server.
203          * @exception       NotBoundException If the server name indicated as the
204            *                                       next destination in the Itinerary cannot be found on the
205            *                                       network.
206          */
207             public static void sendPackage(AgentPackage agentPackage)
208                                 throws NoSuchElementException, IOException,
209                                       RemoteException, NotBoundException{
210
211                 // retrieve a remote reference to the next destination
212                 ConduitServer nextConduit = (ConduitServer)Naming.lookup(
213                                       buildRmiURL(agentPackage.getItinerary().nextDestination().getDestinationHost
))));
214
215                 agentPackage.prepareForTransport();
216
217                 // pass the agent along
218                 nextConduit.receivePackage(agentPackage);
219             }
220
221
222         /**
223          * Receive an Agent from the previous conduit server in its Itinerary.
224          * @param           agentData A stream of bytes containing the serialized
225            *                                       agent.
226          * @param           codebase The agents MobileCodebase
227          * @param           itinerary The agents Itinerary
228          * @exception       RemoteException If an error occurs communicating with
229            *                                       the prior server.
230          */
231         public void receivePackage(AgentPackage agentPackage)
232                                                 throws RemoteException {
233                 Debug.println("conduit", 2, "ConduitServerImpl.receivePackage called");
234                 agentPackage.restoreFromTransportForm();
```

```
                Debug.println("conduit", 1, "Agent " + agentPackage.getAgent().getAgentID() +" has arrived");

                incrementAgentCounters();

                AgentThread newThread = new AgentThread(agentPackage);
        }

    static public void restartAgent(AgentPackage agentPackage) {
                Debug.println("conduit", 1, "Restarting Agent " + agentPackage.getAgent().getAgentID());

                incrementAgentCounters();

                try {
                        if (agentPackage.getItinerary().getCurrentLocation().hasCompleted()) {
                                Debug.println("conduit", 1, "\tAgent had completed execution on this Conduit Server");

                                Debug.println("conduit", 1, "\tSending Agent to next Destination");

                                decrementAgentCounters();
                                sendPackage(agentPackage);

                                // remove the agent from the persistence store
                                getPersistenceStore().removeFromStore(agentPackage);
                        } else {
                                Debug.println("conduit", 1, "\tAgent had not completed execution on this Conduit Server");

                                Debug.println("conduit", 1, "\tRestarting execution");
                                AgentThread newThread = new AgentThread(agentPackage);
                        }
                } catch (Exception e) {
                        ErrorLog.println("conduit.ConduitServerImpl",
                                        ErrorLog.SEVERITY_ERROR,
                                        "An error occurred while restarting agent:",
                                        e);
            }

        }

    /**
         * REtrieves a property from the Conduit Server
         * @param               key The name of the propery to retrieve
         * @param               def A default value which will be returned if
         *                              the property cannot be found
     * @return          the propery value (a string)
         */
        public static String getProperty(String key, String def) {
                return theServer.itsProperties.getProperty(key, def);
```

```
280           }
281
282       /**
283            * Returns the conduit servers name.
284        * @exception       RemoteException If any network error occurs.
285            */
286           public static String getConduitName() throws RemoteException {
287           return theServer.itsName;
288       }
289
290
291           /**
292            * This method is used internally to build a full RMI URL for a conduit
293            * server on the given host.
294            */
295           protected static String buildRmiURL(String hostname) {
296                   return new String("rmi://" + hostname + "/" + RMI_NAME);
297           }
298
299
300       /**
301            * Returns a refernece to the conduit servers persistence store.
302            */
303           public static ConduitServerPersistenceStore getPersistenceStore() {
304           return theServer.itsPersistenceStore;
305       }
306
307       /**
308            * Returns the startup time of this conduit server.
309            */
310           public static Date getStartupTime() {
311           return theServer.itsStartTime;
312       }
313
314       /**
315            * Returns the uptime of this conduit server (in milliseconds).
316            */
317           public static long getUpTime() {
318           return (System.currentTimeMillis() - theServer.itsStartTime.getTime());
319       }
320
321       /**
322            * Returns the total number of agents to pass through this
323            * Conduit Server since it was started.
324            */
325           public static long getTotalAgentCount() {
326           return theServer.itsTotalAgentCount;
```

WO 98/21662 PCT/US97/20232

```
327     }
328
329     /**
330          * Returns the number of agents currently executing
331          * in this Conduit Server.
332          */
333         public static long getCurrentAgentCount() {
334         return theServer.itsCurrentAgentCount;
335     }
336
337     /**
338          * Returns the maximum number of agents to be executing
339          * at any one time in this Conduit Server.
340          */
341         public static long getPeakAgentCount() {
342         return theServer.itsPeakAgentCount;
343     }
344
345
346         /**
347          *
348          */
349         public static synchronized void incrementAgentCounters() {
350                 if (theServer != null) {
351                         theServer.itsTotalAgentCount++;
352                         theServer.itsCurrentAgentCount++;
353                         if (theServer.itsCurrentAgentCount > theServer.itsPeakAgentCount)
354                                 theServer.itsPeakAgentCount =  theServer.itsCurrentAgentCount;
355                 }
356         }
357
358         /**
359          *
360          */
361         public static synchronized void decrementAgentCounters() {
362                 if (theServer != null) {
363                         theServer.itsCurrentAgentCount--;
364                 }
365         }
366
367         /**
368          *  Performs server startup tasks
369          */
370         void startup() {
371                 itsPersistenceStore.restorePersistedObjects();
372
373                 // Monitor memory if conduit.memory facility is set to
```

```
374                     // debug level 4
375                     if (Debug.enabled("conduit.memory.watch", 4)) {
376                             MemoryWatch watch = new MemoryWatch();
377                     }
378
379                     itsStartTime = new Date();
380             }
381
382         /**
383              * A simple bootstrap which brings the conduit server into existence,
384              * registers it name with the RMI naming service and begins listening
385              * for Agent requests.  The main routine allows ConduitServerImpl to
386              * be intiated from the command line via the java command.
387              */
388             public static void main(String args[]) {
389                 // Create and install the security manager
390                 //System.setSecurityManager(new StubSecurityManager());
391                     System.setSecurityManager(new InsecurityManager());
392
393                 try {
394                             // construct the server and register it with the RMI Naming service
395                     Debug.println("conduit", 1, "Create an Conduit Server");
396                             String url = buildRmiURL(InetAddress.getLocalHost().getHostName());
397
398                     ConduitServerImpl server = new ConduitServerImpl(url);
399
400                     Debug.println("conduit", 1, "Bind it to name: " + RMI_NAME);
401                     Naming.rebind(RMI_NAME, server);
402
403                             server.startup();
404                     Debug.println("conduit", 1, "Conduit Server ready.");
405                 } catch (Exception e) {
406                             ErrorLog.println("conduit.main", ErrorLog.SEVERITY_FATAL,
407                                                     "an exception occurred during startup:", e);
408                 }
409         }
410 }
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/conduit/ConduitServerPersistenceStore.java 6     11/11/96 5:15p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Conduit Server persistence Store
 *
 *      $Log: /com/meitca/hsl/zonesjagents/conduit/ConduitServerPersistenceStore.java $
 *
 * 6     11/11/96 5:15p Billp
 * Made correction to company name.
 *
 * 5     10/29/96 5:29p Walsh
 * Use ErrorLog class to report errors
 *
 * 4     10/21/96 4:26p Walsh
 * Add debug logging
 *
 * 3     10/21/96 11:38a Walsh
 * mirror change in Persistence API
 *
 * 2     10/18/96 10:58a Walsh
 * synch with change to findObject API
 *
 * 1     10/10/96 6:01p Walsh
 * initial version
 *
 */
package com.meitca.hsl.zonesjagents.conduit;

import java.io.File;
import java.util.*;

import com.meitca.hsl.zonesjagents.persistence.*;
import com.meitca.hsl.zonesjagents.shared.*;
import com.meitca.hsl.util.*;

/**
 * Conduit Server Persistence Store
 * @see         ConduitServerImpl
 * @author      Thomas Walsh
 */
class ConduitServerPersistenceStore {
```

WO 98/21662 PCT/US97/20232

```
48          // String constants representing the propery names of
49          // two properties of interest
50          private static final String ENABLE_PERSISTENCE = "conduit.persistence.Enable";
51          private static final String PERSISTENCE_FILE = "conduit.persistence.FileName";
52          private static final String DEFAULT_FILENAME = "persist.store";
53
54          /** The PersistentStoreManager object used to manager the persistence store */
55          PersistentStoreManager  itsStore;
56
57          /**
58           * Constructs the PersistenceStore.  Persistence will be enabled only
59           * if the conduit.persistence.Enable Property is set to "true".  The
60           * PersistenceStore will be stored in the file named in the
61           * conduit.persistence.FileName property.  If this Property is not set,
62           * the store will be placed in the file "zones.home/Agent/persist.store"
63           * only
64           */
65          ConduitServerPersistenceStore() {
66                  if (new Boolean(ConduitServerImpl.getProperty(ENABLE_PERSISTENCE, "false")).booleanValue()) {
67                          Debug.println("conduit.persistence", 1, "Persistence enabled.");
68
69                          String filename = ConduitServerImpl.getProperty(PERSISTENCE_FILE, "");
70                          Debug.println("conduit.persistence", 3, "conduit.persistence.FileName property set to " + fi
ename);
71
72                          if (filename.length() == 0) {
73                                  filename = AgentConstants.AGENT_DIR + DEFAULT_FILENAME;
74                          }
75
76                          try {
77                                  // construct the PersistentStoreManager
78                                  Debug.println("conduit.persistence", 2, "Persistence filename is " + filename);
79                                  itsStore = new PersistentStoreManager(filename);
80                          } catch (PersistentStoreException e) {
81                                  ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_ERROR,
82                                                                          "An error occurred constructing the persiste
ce store.", e);
83                                  ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_WARNING,
84                                                                          "Continuing without persistence.");
85
86                                  // null out the itsStore reference.  We will continue
87                                  // executing with persistence disabled
88                                  itsStore = null;
89                          }
90
91                  } else {
92                          // null out the itsStore reference.  This will
```

```
 93                          // disable persistence
 94                          itsStore = null;
 95                  }
 96
 97          }
 98
 99          /**
100           * Adds an AgentPackage to the persistence store.
101           * @param       aPackage The package to add.
102           */
103          void addToStore(AgentPackage aPackage) {
104                  if (itsStore != null) {
105                          if (aPackage.getOID() == 0) {
106                                  Debug.println("conduit.persistence", 3, "Adding agent to persistence store: " + aPa
age);
107
108                                  try {
109                                          int oid = itsStore.insertObject(aPackage);
110                                          Debug.println("conduit.persistence", 4, "Agent's oid is: " + oid);
111                                          aPackage.setOID(oid);
112                                  } catch (PersistentStoreDuplicateObjectException e1) {
113                                          try {
114                                                  Debug.println("conduit.persistence", 4, "Agent already in store tryi
g update");
115
116                                                  int oid = itsStore.findObject(aPackage);
117
118                                                  if (oid != PersistentStoreManager.INVALID_OID) {
                                                            Debug.println("conduit.persistence", 4, "Agent found in stor
oid: " + oid);
119
120                                                          itsStore.updateObject(oid, aPackage);
121                                                          aPackage.setOID(oid);
122                                                  } else {
                                                            ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_ER
OR,
123                                                                         "An error occurred adding agent " + aPackage
getAgent().getAgentID() +
124                                                                         " to the persistence store.");
125
126                                                         ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_ER
OR,
127                                                                         "PersistentStoreManager.findObject returned n
ull even though " +
128                                                                         "PersistentStoreManager.insertObject threw a
" +
129                                                                         "PersistentStoreDuplicateObjectException.");
130
131                                                         ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_WA
```

```
NING,
                                                                "Continuing without persistence.");
                                                        // null out the itsStore reference.  We will continue
                                                        // executing with persistence disabled
                                                        itsStore = null;
                                                }
                                        } catch (PersistentStoreException e2) {
                                                ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_ERROR,
                                                                "An error occurred adding agent " + aPackage.
getAgent().getAgentID() +
                                                                " to the persistence store.", e2);
                                                ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_WARNING,
                                                                "Continuing without persistence.");
                                                // null out the itsStore reference.  We will continue
                                                // executing with persistence disabled
                                                itsStore = null;
                                        }
                                } catch (PersistentStoreException e3) {
                                        ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_ERROR,
                                                        "An error occurred adding agent " + aPackage.getAgent
().getAgentID() +
                                                        " to the persistence store.", e3);
                                        ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_WARNING,
                                                        "Continuing without persistence.");
                                        // null out the itsStore reference.  We will continue
                                        // executing with persistence disabled
                                        itsStore = null;
                                }
                        }
                }
        }

        /**
         * Updates an AgentPackage in the persistence store.
         * @param       aPackage The package to add.
         */
        void updateStore(AgentPackage aPackage) {
                if (itsStore != null) {
                        Debug.println("conduit.persistence", 3, "Updating agent in persistence store: " + aPackage);
                        Debug.println("conduit.persistence", 3, "Agent's oid is : " + aPackage.getOID());

                        try {
```

WO 98/21662 PCT/US97/20232

```
176                                 itsStore.updateObject(aPackage.getOID(), aPackage);
177                             } catch (PersistentStoreException e2) {
178                                 ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_ERROR,
179                                                 "An error occurred updating agent " + aPackage.getAgent().ge
AgentID() +
180                                                 " in the persistence store.", e2);
181
182                                 ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_WARNING,
183                                                 "Continuing without persistence.");
184
185                                 // null out the itsStore reference.  We will continue
186                                 // executing with persistence disabled
187                                 itsStore = null;
188                             }
189                     }
190
191             }
192
193             /**
194              * Removes an AgentPackage to the persistence store.
195              * @param       aPackage The package to remove.
196              */
197             void removeFromStore(AgentPackage aPackage) {
198                     if (itsStore != null) {
199                             Debug.println("conduit.persistence", 3, "Removing agent from persistence store: " + aPackage
200                             Debug.println("conduit.persistence", 3, "Agent's oid is : " + aPackage.getOID());
201
202                             try {
203                                     itsStore.deleteObject(aPackage.getOID());
204                             } catch (PersistentStoreException e) {
205                                     ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_ERROR,
206                                                     "An error occurred removing agent " + aPackage.getAgent().ge
AgentID() +
207                                                     " from the persistence store.", e);
208
209                                     ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_WARNING,
210                                                     "Continuing without persistence.");
211
212                                     // null out the itsStore reference.  We will continue
213                                     // executing with persistence disabled
214                                     itsStore = null;
215                             }
216                     }
217             }
218
219             /**
```

```
220          * Restores all Agents in the Persistence Store and
221          * passes them into the ConduitServerImpl.restartAgent method
222          * so that they are restarted.
223          */
224         void restorePersistedObjects() {
225                 if (itsStore != null) {
226                         Debug.println("conduit.persistence", 3, "Restoring all persisted Agents");
227
228                         Enumeration enum = itsStore.getOIDs();
229
230                         while (enum.hasMoreElements()) {
231                                 ObjectID objectID = (ObjectID)enum.nextElement();
232                                 int oid = objectID.getOID();
233
234                                 Debug.println("conduit.persistence", 4, "Restoring Agent.   Oid is : " + oid);
235
236                                 if (oid != 0) {
237                                         try {
238                                                 AgentPackage aPackage = (AgentPackage)itsStore.fetchObject(oid);
239                                                 Debug.println("conduit.persistence", 4, "Agent Package is : " + aPac
age);
240
241                                                 aPackage.setOID(oid);
242                                                 ConduitServerImpl.restartAgent(aPackage);
243                                         } catch (PersistentStoreException e) {
244                                                 ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_ERROR,
245                                                                 "An error occurred restoring persisted Agents
", e);
246
247                                                 ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_WARNING,
248                                                                 "Continuing without persistence.");
249
250                                                 // null out the itsStore reference.  We will continue
251                                                 // executing with persistence disabled.
252                                                 itsStore = null;
253                                         }
254                                 }
255
256                         }
257                 }
258         }
```

WO 98/21662 PCT/US97/20232

```
/*  $Header: /com/meitca/hsl/zonesjagents/conduit/Destination.java 7     11/11/96 5:15p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  An agents destination.  Contains RMI URL of conduit server, and
 *  method id of method to invoke at that location
 *
 *      $Log: /com/meitca/hsl/zonesjagents/conduit/Destination.java $
 *
 * 7     11/11/96 5:15p Billp
 * Made correction to company name.
 *
 * 6     10/10/96 6:02p Walsh
 * Add persistence support
 *
 * 5     9/30/96 7:06p Walsh
 * clean up javadoc comments
 *
 * 4     9/29/96 4:08p Walsh
 * Allow Destination methods to be specified by name
 * Give Agent access to AgentPackage information
 *
 * 3     9/04/96 12:31p Walsh
 * Destination now holds only the hostname of the ConduitServer, not the
 * entire RMI URL
 *
 * 2     8/29/96 5:16p Walsh
 * add default constructor and set* methods
 *
 * 1     8/23/96 3:50p Walsh
 * Initial versions
 */
package com.meitca.hsl.zonesjagents.conduit;

import com.meitca.hsl.zonesjagents.conduit.InvalidMethodNameException;

/**
 * An object representing a Destination in an Agents Itinerary.  The
 * Destination is composed of two parts; <br>
 *              <ol>
 *                      <li>the hostname of the  conduit server in which to travel
 *                      <li> An identifer indicating the method of the agent to call
```

WO 98/21662 PCT/US97/20232

```
48   *                                at that ConduitServer.  This identifier can either be an
49   *                                number representing a (zero based) ID of the method or the
50   *                                name of the method in the form of a String.
51   *              </ol>
52   * @see         Agent
53   * @see         ConduitServer
54   * @see         Itinerary
55   * @author      Thomas Walsh
56   */
57  public class Destination implements Cloneable {
58          static final int UNKNOWN_METHOD_ID = -1;
59
60          /** The hostname of the Conduit Server in which to travel */
61          String  itsHostName;
62
63          /** The method ID number of the method to invoke */
64          int             itsMethodID;
65
66          /** The name of the method to invoke */
67          String  itsMethodName;
68
69          /**
70           * A boolean indicating wether execution at this destination
71           * has completed.
72           */
73          boolean itsHasCompleted;
74
75          /**
76           * Constructs a Destination object.
77           * @param       hostname The hostname of the conduit server in which to
78           *                       travel
79           * @param       methodName  The name of the method to invoke
80           */
81          public Destination(String hostname, String methodName) {
82                  itsHostName = hostname;
83                  itsMethodID = UNKNOWN_METHOD_ID;
84                  itsMethodName = methodName;
85                  itsHasCompleted = false;
86          }
87
88
89          /**
90           * Constructs a Destination object.  This constructor is used
91           * internally to construct a Destination.
92           * @param       hostname The hostname of the conduit server in which to
93           *                       travel
94           * @param       method The method ID number of the method to invoke
```

```
 95          */
 96         public Destination(String hostname, int method) {
 97                 itsHostName = hostname;
 98                 itsMethodID = method;
 99                 itsMethodName = null;
100                 itsHasCompleted = false;
101         }
102
103         /**
104          * Constructs an empty Destination object
105          */
106         public Destination() {
107                 itsHostName = new String("");
108                 itsMethodID = UNKNOWN_METHOD_ID;
109                 itsMethodName = null;
110                 itsHasCompleted = false;
111         }
112
113         /**
114          * Retrieves the hostname of the conduit server in which to travel
115          */
116         public String getDestinationHost() {
117                 return itsHostName;
118         }
119
120         /**
121          * Retrieves the method ID number of the method to invoke
122          */
123         public int getMethodID() {
124                 return itsMethodID;
125         }
126
127         /**
128          * Sets the URL of the conduit server in which to travel
129          */
130         public void setDestinationHost(String hostname) {
131                 itsHostName = hostname;
132         }
133
134         /**
135          * Sets the method ID number of the method to invoke
136          */
137         public void setMethodID(int id) {
138                 itsMethodName = null;
139                 itsMethodID = id;
140         }
141
```

WO 98/21662 PCT/US97/20232

```
142         /**
143          * Retrieves the method name of the method to invoke
144          */
145         public String getMethodName() {
146                 return itsMethodName;
147         }
148
149         /**
150          * Makes a clone of the Destination object
151          */
152         public Destination cloneDestination() {
153                 Destination clone = new Destination();
154                 clone.itsHostName = itsHostName ;
155                 clone.itsMethodID = itsMethodID ;
156                 clone.itsMethodName = itsMethodName;
157                 return clone;
158         }
159
160         /**
161          * Sets the internal state of the Destination to
162          * indicate that execution has been completed.
163          */
164         void setCompleted() {
165                 itsHasCompleted = true;
166         }
167
168         /**
169          * Checks the internal state of the Destination to
170          * see if execution has been completed.
171          */
172         boolean hasCompleted() {
173                 return itsHasCompleted;
174         }
175
176     /** Used internally to convert a method name into a method id. */
177         protected void convertNameToId(AgentSkeleton skel)
178                                                 throws InvalidMethodNameException {
179
180                 if (itsMethodName == null) {
181                         throw new InvalidMethodNameException("No name specified for destination");
182                 }
183
184                 String[] methods = skel.getMethods();
185                 for(int i=0; i<methods.length; i++) {
186                         if (itsMethodName.equals(methods[i])) {
187                                 itsMethodID = i;
188                                 itsMethodName = null;
```

```
   189                                  return;
   190                              }
   191                          }
   192                          throw new InvalidMethodNameException("Method name " + itsMethodName + "does not match any methods of
agent");
   193                  }
   194          }
```

```
1  /*      $Header: /ccm/meitca/hsl/zonesjagents/conduit/InvalidMethodNameException.java 3     11/11/96 5:15p Billp $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory,
4   *  Mitsubishi Electric Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  Exception indicating that Method Name speicifed in a Destination
11  *  is not a valid name
12  *
13  *      $Log: /com/meitca/hsl/zonesjagents/conduit/InvalidMethodNameException.java $
14  *
15  * 3     11/11/96 5:15p Billp
16  * Made correction to company name.
17  *
18  * 2     9/30/96 7:06p Walsh
19  * clean up javadoc comments
20  *
21  * 1     9/29/96 4:07p Walsh
22  * initial version
23  *
24  */
25 package com.meitca.hsl.zonesjagents.conduit;
26
27 /**
28  * An exception which indicates that the method name that was specified
29  * in a Destination was not the name of a method of the agent which
30  * is travelling.
31  * @see         Agent
32  * @see         AgentSkeleton
33  * @author      Thomas Walsh
34  */
35 public class InvalidMethodNameException extends Exception {
36     public InvalidMethodNameException() {
37                 super();
38     }
39
40     public InvalidMethodNameException(String s) {
41                 super(s);
42     }
43 }
```

WO 98/21662 PCT/US97/20232

```
 1  /*  $Header: /com/meitca/hsl/zonesjagents/conduit/Itinerary.java 5     11/11/96 5:15p Billp $
 2   *
 3   *  Copyright  1996 Horizon Systems Laboratory,
 4   *  Mitsubishi Electric Information Technology Center America.
 5   *      All rights reserved.
 6   *
 7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 8   *
 9   *  DESCRIPTION
10   *  Agent itinerary object
11   *
12   *      $Log: /com/meitca/hsl/zonesjagents/conduit/Itinerary.java $
13   *
14   * 5     11/11/96 5:15p Billp
15   * Made correction to company name.
16   *
17   * 4     9/29/96 4:08p Walsh
18   * Allow Destination methods to be specified by name
19   * Give Agent access to AgentPackage information
20   *
21   * 3     8/23/96 3:58p Walsh
22   * Add support for ad-hoc method invocation
23   *
24   * 2     8/09/96 4:38p Walsh
25   * Setting up file headers
26   */
27  package com.meitca.hsl.zonesjagents.conduit;
28
29  import java.util.*;
30
31  /**
32   * An object representing an Agent's Itinerary.  The Itinerary
33   * indicated WHERE an agent should travel and indicated WHAT
34   * the Agent should do at each stop
35   * @see         Agent
36   * @see         ConduitServer
37   * @see         Destination
38   * @author      Thomas Walsh
39   */
40  public class Itinerary {
41      Vector  itsDestinations;
42      int     itsCurrentStop;
43
44      /**
45           * Constructs an Itinerary object
46           */
47          public Itinerary() {
```

WO 98/21662 PCT/US97/20232

```
48          itsDestinations = new Vector();
49          itsCurrentStop = -1;
50      }
51
52      /**
53           * Adds a new destination to the end of the Agents Itinerary.
54           * @param       destination The new destination
55           */
56      public void addDestination(Destination destination) {
57          itsDestinations.addElement(destination);
58      }
59
60          /**
61           * Clears all destinations from the Agents Itinerary
62           */
63          public void clearItinerary() {
64                  itsDestinations.removeAllElements();
65          }
66
67          /**
68           * Resets the Itinerary back to the initial Destination.
69           * This method could be used to force an Agent to restart
70           * its Itinerary and revisit all Destinations
71           */
72          public void resetItinerary() {
73          itsCurrentStop = -1;
74          }
75
76
77       /**
78            * Retrieves the destination object identifying the current
79            * location of the Agent
80            * @returns              The Agent's current location
81            * @exception   NoSuchElementException If the Agent has not
82            *                                  travelled.
83            */
84          public Destination getCurrentLocation() throws NoSuchElementException {
85          try {
86              return (Destination)itsDestinations.elementAt(itsCurrentStop);
87          } catch (ArrayIndexOutOfBoundsException boundError) {
88                          throw new NoSuchElementException();
89          }
90          }
91
92       /**
93            * Retrieves the destination object identifying the current
94            * location of the Agent
```

```
 95          * @returns             The Agent's next destination
 96          * @exception   NoSuchElementException If the Agent has not
 97          *                                  travelled.
 98          */
 99         public Destination getNextDestination() throws NoSuchElementException {
100         try {
101             return (Destination)itsDestinations.elementAt(itsCurrentStop+1);
102         } catch (ArrayIndexOutOfBoundsException boundError) {
103                     throw new NoSuchElementException();
104         }
105         }
106
107         /**
108          * Retrieves an enumeration of all of the Destinations in the
109          * Itinerary.
110          * @returns             An enumeration of all of the Destinations in the
111          *                              Itinerary.
112          */
113         public Enumeration destinations() {
114                 return itsDestinations.elements();
115         }
116
117         /**
118          * Makes a clone of the Itinerary.  If the reset parameter is true
119          * then the new Itinerary is reset to the first destination.  If
120          * reset is false, then the new itinerary is set such that its next
121          * destination is the same as that of the original
122          * @param       reset true if the new itinerary should be reset to
123          *                      the first destination.
124          * @returns A clone of the original Itinerary
125          */
126         public Itinerary cloneItinerary(boolean reset) {
127                 Itinerary clone = new Itinerary();
128
129                 Enumeration enum = itsDestinations.elements();
130                 while (enum.hasMoreElements()) {
131                         Destination original = (Destination)enum.nextElement();
132                         Destination cloneDest = (Destination)original.cloneDestination();
133                         clone.addDestination(cloneDest);
134                 }
135
136                 if (!reset)
137                         clone.itsCurrentStop = itsCurrentStop;
138
139                 return clone;
140         }
141
```

```
142         /**
143          * Retrieves the next destination to which an Agent should
144          * travel and increments the Itineraries internal state to
145          * the next destination.  This method is called internally
146          * by the ConduitServer when it is preparing to transport
147          * the Agent to its next destination.
148          */
149         protected Destination nextDestination() throws NoSuchElementException {
150         try {
151             return (Destination)itsDestinations.elementAt(++itsCurrentStop);
152
153         } catch (ArrayIndexOutOfBoundsException boundError) {
154             throw new NoSuchElementException();
155         }
156     }
157 }
```

```
1  /*  $Header: /com/meitca/hsl/zonesjagents/conduit/MemoryWatch.java 3     11/11/96 5:15p Billp $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory,
4   *  Mitsubishi Electric Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  Conduit Server implementation
11  *
12  *      $Log: /com/meitca/hsl/zonesjagents/conduit/MemoryWatch.java $
13  *
14  * 3     11/11/96 5:15p Billp
15  * Made correction to company name.
16  *
17  * 2     10/28/96 6:31p Walsh
18  * Add more debugging info
19  *
20  * 1     10/24/96 5:12p Walsh
21  * Initial Version
22  *
23  */
24 package com.meitca.hsl.zonesjagents.conduit;
25
26 import java.util.*;
27
28 import com.meitca.hsl.util.*;
29
30 /**
31  * The memory watch class provides an minimum priority thread which
32  * reports the total amount of memory allocated to the Java VM as
33  * well as the amount of free memory currently available.
34  *
35  * To enable the MemoryWatch, the Conduit Server should be started
36  * up with the following switch passed into java.exe
37  *              -DDebug="conduit.memory.watch=4"
38  * @author      Thomas Walsh
39  */
40 class MemoryWatch extends Thread {
41
42         /** The number of milliseconds between polls of the VM */
43         static int pollingInterval = 5000; // in millisec
44
45         /** Constructs a MemoryWatch object */
46         MemoryWatch() {
47                 Debug.println("conduit.memory.watch", 1, "Memory Watch thread starting...");
```

```
48                  setPriority(Thread.MIN_PRIORITY);
49              start();
50              }
51
52              /** Performs memory polling */
53              public void run() {
54                      while (true) {
55                              Debug.println("conduit.memory.watch", 4, "");
56                              Debug.println("conduit.memory.watch", 4, "Server up since: " + ConduitServerImpl.getStartup
me());
57
58                              long uptime = ConduitServerImpl.getUpTime();
59                              Date up_date = new Date(uptime);
60                              Date bigBang = new Date(0);
61                              Debug.println("conduit.memory.watch", 4, "Server uptime: " +
62                                                      (up_date.getYear() - bigBang.getYear()) + " years, " +
63                                                      (up_date.getMonth() - bigBang.getMonth()) + " months, " +
64                                                      (up_date.getDate() - bigBang.getDate()) + " days, " +
65                                                      (up_date.getHours() - bigBang.getHours()) + " hours, " +
66                                                      (up_date.getMinutes() - bigBang.getMinutes()) + " minutes, "
+
67                                                      (up_date.getSeconds() - bigBang.getSeconds()) + " seconds. ")
;
68
69                              Debug.println("conduit.memory.watch", 4, "Total number of Agents:\t\t\t" + ConduitServerImpl.
getTotalAgentCount());
70                              Debug.println("conduit.memory.watch", 4, "Current number of Agents:\t\t" + ConduitServerImpl.
getCurrentAgentCount());
71                              Debug.println("conduit.memory.watch", 4, "Peak number of Agents:\t\t\t" + ConduitServerImpl.
etPeakAgentCount());
72                              Debug.println("conduit.memory.watch", 4, "Total memory allocated to system:\t" + Runtime.ge
untime().totalMemory());
73                              Debug.println("conduit.memory.watch", 4, "Total free memory available:\t\t" + Runtime.getRun
ime().freeMemory());
74                              Debug.println("conduit.memory.watch", 4, "");
75                              try {
76                                      sleep(pollingInterval);
77                              } catch (InterruptedException e) {
78                                      Debug.println("conduit.memory.watch", 4, "Some thread interrupted the MemoryWatch du
ing sleep.  Continuing");
79                              }
80                      }
81              }
82
83  }
```

WO 98/21662 PCT/US97/20232

```
 1  /*  $Header: /com/meitca/hsl/zonesjagents/conduit/MobileCodebase.java 5     11/11/96 5:15p Billp $
 2   *
 3   *  Copyright  1996 Horizon Systems Laboratory,
 4   *  Mitsubishi Electric Information Technology Center America.
 5   *      All rights reserved.
 6   *
 7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 8   *
 9   *  DESCRIPTION
10   *  Agent Mobile codebase object
11   *
12   *      $Log: /com/meitca/hsl/zonesjagents/conduit/MobileCodebase.java $
13   *
14   * 5     11/11/96 5:15p Billp
15   * Made correction to company name.
16   *
17   * 4     9/30/96 7:06p Walsh
18   * clean up javadoc comments
19   *
20   * 3     8/13/96 11:18a Walsh
21   * Add documentation & comments.  Set up use of HashTable for bytecode
22   * storage.
23   *
24   * 2     8/09/96 4:38p Walsh
25   * Setting up file headers
26   */
27  package com.meitca.hsl.zonesjagents.conduit;
28
29  import java.util.Hashtable;
30
31
32  /**
33   * Contains the bytecodes for the objects that travel with an Agent.
34   * @see         Agent
35   * @author      Thomas Walsh
36   */
37  public class MobileCodebase {
38      /**
39           * Provides for easy storage and retrieval of bytecodes.   Bytecodes
40           * are stored in a Hashtable and are accessible by fully qualified
41           * Class name (ie "java.lang.String").
42           */
43          Hashtable itsByteCodes;
44
45      /**
46           * Constructs a MobileCodebase object
47           */
```

WO 98/21662 PCT/US97/20232

```
48          public MobileCodebase() {
49                  itsByteCodes = new Hashtable();
50      }
51
52          /**
53           * Stores an objects bytecodes into the MobileCodebase.
54       * @param          name The fully qualified Class name (ie
55           *                              <b>"java.lang.String"</b>).
56       * @param          code The Class's bytecodes
57           */
58      public void storeCode(String name, byte[] code) {
59                  itsByteCodes.put(name, code);
60      }
61
62          /**
63           * Retrieves an objects bytecodes from the MobileCodebase.
64       * @param          name The fully qualified Class name (ie
65           *                              <b>"java.lang.String"</b>).
66           */
67      public byte[] getCode(String name) {
68          return (byte[]) itsByteCodes.get(name);
69      }
70  }
```

```
 1  /*  $Header: /com/meitca/hsl/zonesjagents/remoteloader/RemoteClassLoader.java 4     11/11/96 5:15p Billp
 2   *
 3   *  Copyright  1996 Horizon Systems Laboratory,
 4   *  Mitsubishi Electric Information Technology Center America.
 5   *      All rights reserved.
 6   *
 7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 8   *
 9   *  DESCRIPTION
10   *  Remote class loader distributed interface
11   *
12   *      $Log: /com/meitca/hsl/zonesjagents/remoteloader/RemoteClassLoader.java $
13   *
14   * 4     11/11/96 5:15p Billp
15   * Made correction to company name
16   *
17   * 3     9/28/96 6:39p Walsh
18   * Add new parameter to retrieveClass method
19   *
20   * 2     8/27/96 4:10p Walsh
21   * retrieveClass throws ClassNotFoundException (compiler games)
22   *
23   * 1     8/19/96 5:28p Walsh
24   * Initial version
25   */
26  package com.meitca.hsl.zonesjagents.remoteloader;
27
28  import java.rmi.Remcte;
29  import java.rmi.RemcteException;
30
31
32  /**
33   * RemoteClassLoader is an RMI distributed interface.  It provides a
34   * mechanism by which conduit servers can retrieve bytecodes for classes
35   * that exist on machines other than the one on which the conduit
36   * server is loading.
37   * @see         ConduitServer
38   * @author      Thomas Walsh
39   */
40  public interface RemoteClassLoader extends Remote {
41
42      /**
43       * Retrieve the bytecodes for the specified class.
44       * @param           classURL An URL specifying the class to be loaded.
45            *                                This should be of the form /directory_path/classname
46            * @return              The bytecodes for the class.
47            * @exception   RemoteException If an error occurs communicating.
```

```
48        */
49       public byte[] retrieveClass(String codebaseURL, String className) throws RemoteException,
                                                                                   ClassNotFoundException;
50
51   }
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/remoteloader/RemoteClassLoaderImpl.java 8     11/11/96 5:15p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Remote class loader implementation
 *
 *      $Log: /com/meitca/hsl/zonesjagents/remoteloader/RemoteClassLoaderImpl.java $
 *
 * 8     11/11/96 5:15p Billp
 * Made correction to company name
 *
 * 7     10/29/96 5:31p Walsh
 * Use ErrorLog class to report errors
 *
 * 6     10/21/96 1:26p Walsh
 * Add some debug output
 *
 * 5     9/28/96 6:40p Walsh
 * Use AgentCodebase class to load byte codes.
 * Add retrieveRemoteClass to encapsulate RMI code
 *
 * 4     9/04/96 12:33p Walsh
 * Centralize manipulation or RMI URL.  Use the test security manager
 *
 * 3     8/27/96 4:18p Walsh
 * Add silly constructor (because Sun compiler wants it)
 *
 * 2     8/23/96 3:59p Walsh
 * Add logging message
 *
 * 1     8/19/96 5:28p Walsh
 * Initial version
 */
package com.meitca.hsl.zonesjagents.remoteloader;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteServer;
import java.rmi.server.StubSecurityManager;
import java.rmi.Naming;
import java.io.*;
import java.lang.ClassNotFoundException;
```

```
48
49
50  import com.meitca.hsl.zonesjagents.security.*;
51  import com.meitca.hsl.zonesjagents.shared.*;
52  import com.meitca.hsl.util.*;
53
54  /**
55   * An RMI distributed object which provides for remote retrieval of class
56   * bytecodes. RemoteClassLoaderImpl implements the RemoteClassLoader interface.
57   * @see         RemoteClassLoader
58   * @author      Thomas Walsh
59   */
60  public class RemoteClassLoaderImpl
61                                          extends UnicastRemoteServer
62                          implements RemoteClassLoader {
63
64          /**The name to which a conduit server is bound in the RMI Registry */
65          static String                                   RMI_NAME = "RemoteClassLoader";
66
67          /**
68           * object which actually handles the dirty work or retrieving .class
69           * files for a given class.
70           */
71          ClassFileLoader                                 itsClassFileLoader;
72
73
74          public RemoteClassLoaderImpl()  throws RemoteException {
75                  itsClassFileLoader = new ClassFileLoader();
76          }
77
78      /**
79       * Retrieve the bytecodes for the specified class.  This method is invoked
80            * by RMI.  The calling program on the other side of the RMI connection
81            * usually invokes the RMI by calling the
82            * RemoteClassLoader.retrieveRemoteClass method below.
83       * @param           classURL An URL specifying the class to be loaded.
84            *                          This should be of the form /directory_path/classname
85            * @return                 The bytecodes for the class.
86            * @exception   RemoteException If an error occurs communicating.
87       */
88      public byte[] retrieveClass(String codebase_dir, String className)
89                                                      throws RemoteException, ClassNotFoundException {
90
91                  Debug.println("remoteloader", 2, "loading " +
92                          className + " from " + codebase_dir);
93
94                  if ((codebase_dir == null) ||  (codebase_dir.length() == 0)) {
                            Debug.println("remoteloader", 3, "loading from CLASSPATH");
```

```
95                      return itsClassFileLoader.loadClassFileFromClassPath(className);
96              } else {
97                      Debug.println("remoteloader", 3, "loading from directory");
98                      return itsClassFileLoader.loadClassFileFromDirectory(codebase_dir, className);
99              }
100         }
101
102     /**
103      * Retrieve the bytecodes for the specified class from the remote
104         * codebase specified by codebase_URL.  This method is called internally
105         * by the Java agent system whenever an object wants to retrieve code
106         * from a RemoteClassLoader.  This method hides the details of the RMI.
107      * @param           codebase_URL An URL specifying the class to be loaded.
108         *                               This should be of the form /directory_path/classname
109      * @param           className The fully qualified class name of the class whose
110         *                               bytecode should be retrieved.
111         * @return                  The bytecodes for the class.
112         * @exception   RemoteException If an error occurs communicating.
113      */
114        public static byte[] retrieveRemoteClass(String codebase_URL, String className)
115                                                                throws ClassNotFoundException {
116
117                Debug.println("remoteloader", 2, "retrieveRemoteClass: loading " +
118                        className + " from " + codebase_URL);
119
120                int count = 0;
121                int index = 0;
122                while (count < 4) {
123                        index = codebase_URL.indexOf('/', index);
124                        if (index == -1)
125                                break;
126
127                        count++;
128                        index++;
129                }
130
131                String rmiURL;
132                String code_dir;
133
134                if (index != -1) {
135                        rmiURL = codebase_URL.substring(0, (index - 1));
136                        code_dir = codebase_URL.substring(index, codebase_URL.length());
137                } else {
138                        rmiURL = codebase_URL;
139                        code_dir = null;
140                }
141
```

```
142                 Debug.println("remoteloader", 3, "remote object URL: " +  rmiURL);
143                 Debug.println("remoteloader", 3, "remote code dir : " +  code_dir);
144
145                 try {
146                         RemoteClassLoader remote = (RemoteClassLoader)Naming.lookup(rmiURL);
147
148                         return remote.retrieveClass(code_dir, className);
149
150                 } catch (Exception e) {
151                         throw new ClassNotFoundException("Cound not load " + className + e.getMessage());
152                 }
153         }
154
155
156         /**
157          * This method is used internally to build a full RMI URL for a remote
158          * class loader on the given host.
159          */
160         public static String buildRmiURL(String hostname) {
161                 return new String("rmi://" + hostname + "/" + RMI_NAME);
162         }
163
164         /**
165          * This method is used internally to build a full RMI URL for a remote
166          * codebase on the given host.
167          */
168         public static String buildCodebaseURL(String hostname, String codeDir) {
169                 return new String(buildRmiURL(hostname) + "/" + codeDir);
170         }
171
172         /**
173          * A simple bootstrap which brings the remote class loader into existence,
174          * registers it name with the RMI naming service and begins listening
175          * for requests.  The main routine allows RemoteClassLoaderImpl to
176          * be intiated from the command line via the java command.
177          */
178         public static void main(String args[]) {
179             // Create and install the security manager
180             System.setSecurityManager(new InsecurityManager());
181
182             try {
183                         // construct the server and register it with the RMI Naming service
184                         Debug.println("remoteloader", 1, "Create an RemoteClassLoader");
185                 RemoteClassLoaderImpl loader = new RemoteClassLoaderImpl();
186
187                 Debug.println("remoteloader", 1, "Bind it to name: " + RMI_NAME);
188                 Naming.rebind(RMI_NAME, loader);
```

```
189
190                 Debug.println("remoteloader", 1,"RemoteClassLoaderImpl ready.");
191             } catch (Exception e) {
192                     ErrorLog.println("remoteloader.main", ErrorLog.SEVERITY_FATAL,
193                                                 "an exception occurred during startup:", e);
194             }
195     }
196
197 }
```

```
/*      $Header: /ccm/meitca/hsl/zonesjagents/skeleton/Generator.java 5     11/11/96 5:16p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Agent Skeleton Generator
 *
 *      $Log: /com/meitca/hsl/zonesjagents/skeleton/Generator.java $
 *
 * 5     11/11/96 5:16p Billp
 * Made correction to ccmpany name
 *
 * 4     9/30/96 7:20p Walsh
 * add javadoc comments
 *
 * 3     9/29/96 4:32p Walsh
 * Allow agents to be in packages
 *
 * 2     9/12/96 11:00a Walsh
 * Automatically compile the skeleton file
 *
 * 1     9/06/96 4:54p Walsh
 *
 */
package com.meitca.hsl.zonesjagents.skeleton;

import sun.tools.java.*;
import sun.tools.javac.*;
import java.io.*;
import java.util.*;


/**
 * The Generator class generates an Method Invocation Skelton for an
 * Agent.  A Method Invocation Skelton allows the ConduitServer to
 * dynamically invoke any method of an Agent.  The Skeleton serves a
 * similar purpose as does an RMI skeleton or an RPC server side stub.
 * Before any Agent can travel, it is requires that a Skeleton be
 * generated by it.  A programmer can generate a skeleton by executing
 * Generator with a command line like the following: <br>
 *<pre>
 * java com.meitca.hsl.zonesjagents.skeleton.Generator agentclassname
 * </pre>
```

```
48   * The agentclassname paramter should be the fully qualified class
49   * name of the agent.
50   * @see          ConduitServer
51   * @see          Agent
52   * @see          AgentSkeleton
53   * @author       Thomas Walsh
54   */
55  public class Generator {
56          /**
57           * the main method allows Generator to be called from the command
58           * line.
59           */
60          public static void main(String[] args) {
61                  // Sanity check the command line arguments...
62                  if (args.length < 1) {
63                          System.err.println(StringResources.NO_AGENT);
64                          System.err.println(StringResources.USAGE);
65                          return;
66                  }
67
68                  if (args.length > 1) {
69                          System.err.println(StringResources.TOO_MANY);
70                          System.err.println(StringResources.USAGE);
71                          return;
72                  }
73
74                  String agentClassName = args[0];
75                  String packageName;
76                  String shortClassName;
77
78                  int lastSep = agentClassName.lastIndexOf('.');
79                  if (lastSep != -1) {
80                          packageName = agentClassName.substring(0, lastSep);
81                          shortClassName = agentClassName.substring(lastSep + 1, agentClassName.length());
82                  } else {
83                          packageName = null;
84                          shortClassName =  agentClassName;
85
86                  }
87
88                  String skelFileName = shortClassName + "_Skel.java";
89
90
91                  try {
92                          String[] methods = findAgentMethods(agentClassName);
93                          PrintStream out = new PrintStream(new FileOutputStream(skelFileName));
94
```

```
95                      generateSkel(agentClassName, packageName, shortClassName, methods, out);
96                      compileSkel(skelFileName);
97              } catch (ClassNotFound e1) {
98                      System.err.println(StringResources.NOT_FOUND + agentClassName);
99              } catch (IOException e2) {
100                     System.err.println(StringResources.IO_ERROR + skelFileName);
101                     System.err.println(e2.getMessage());
102                     e2.printStackTrace();
103             }
104     }
105
106     static void generateSkel(String agentClassName,
107                                                     String packageName,
108                                                     String shortClassName,
109                                                     String[] methods,
110                                                     PrintStream stream) {
111
112             // Ok.  This is going to be a fairly brute force approach
113             // First lets put a comment on the file
114             stream.println("//Skeleton for " + agentClassName);
115             stream.println("//Generated by com.meitca.hsl.zonesjagents.skeleton.Generator");
116
117             // The package statement
118             if (packageName != null)
119                     stream.println("package " + packageName + ";");
120
121             stream.println("");
122
123             // Now some imports
124             stream.println("import java.lang.IllegalAccessException;");
125             stream.println("import com.meitca.hsl.zonesjagents.shared.Agent;");
126             stream.println("import com.meitca.hsl.zonesjagents.conduit.AgentSkeleton;");
127             stream.println("import " + agentClassName + ";");
128             stream.println("");
129
130             //now the class itself
131             stream.println("public final class " + shortClassName + "_Skel implements AgentSkeleton {")
132
133
134
135             // define the static member variable containing the names of
136             // the objects methods
137             String methodsMember = "\tstatic public String methods[] = { ";
138             for (int i=0; i<methods.length; i++) {
139                     if (i != 0) {
140                             methodsMember = methodsMember + ", ";
141
```

```
142                         }
143                         methodsMember = methodsMember + "new String(\"" + methods[i] + "\")";
144                 }
145                 methodsMember = methodsMember + " };";
146                 stream.println(methodsMember);
147                 stream.println("");
148
149
150                 // now the invoke method
151                 stream.println("\tpublic void invoke(Agent agent, int i) throws IllegalAccessException (");
152                 stream.println("\t\tswitch (i) {");
153                 for (int i=0; i<methods.length; i++) {
154                         stream.println("\t\t\tcase " + i + ":");
155                         stream.println("\t\t\t\t((" + shortClassName + ")agent)." + methods[i] + "();");
156                         stream.println("\t\t\t\tbreak;");
157                 }
158         stream.println("\t\t\tdefault:");
159         stream.println("\t\t\t\tthrow new IllegalAccessException(\"Method \" + i + \" out of range.\");");
160
161                 // close the switch
162                 stream.println("\t\t}");
163
164                 // close the invoke method
165                 stream.println("\t}");
166                 stream.println("");
167
168                 // now the getMethods method
169                 stream.println("\tpublic String[] getMethods() {");
170                 stream.println("\t\treturn methods;");
171                 // close the getMethods method
172                 stream.println("\t}");
173
174                 // close the class definition
175                 stream.println("}");
176         }
177
178         static String[] findAgentMethods(String agentClassName) throws ClassNotFound {
179                 ClassDeclaration        classDecl = new ClassDeclaration(Identifier.lookup(agentClassName));
180                 BatchEnvironment        env = new BatchEnvironment(new ClassPath(System.getProperty("java.class.pat
))));
181                 ClassDefinition         classDef = classDecl.getClassDefinition(env);
182                 Vector                          agentMethods = new Vector();
183
184                 FieldDefinition         fieldDef = classDef.getFirstField();
185                 while (fieldDef != null) {
186                         if (fieldDef.isMethod() && !fieldDef.isConstructor() && fieldDef.isPublic()) {
187                                 Type argTypes[] = fieldDef.getType().getArgumentTypes();
```

WO 98/21662 PCT/US97/20232

```
188                                     if (argTypes.length == 0) {
189                                             if (fieldDef.getType().getReturnType() == Type.tVoid) {
190                                                     agentMethods.addElement(fieldDef.getName().toString());
191                                             }
192                                     }
193                             }
194                             fieldDef = fieldDef.getNextField();
195                     }
196
197                     //turn it into an array
198                     String[] array = new String[agentMethods.size()];
199                     for (int i=0; i<agentMethods.size(); i++) {
200                             array[i] = (String)agentMethods.elementAt(i);
201                     }
202
203                     return array;
204             }
205
206             static void compileSkel(String skelFileName) {
207              Main compiler = new Main(System.out, "javac");
208              String[] params = new String[1];
209              params[0] = skelFileName;
210              if (!compiler.compile(params)) {
211                System.err.println(StringResources.COMPILE_ERROR + skelFileName);
212                }
213             }
214     }
```

```
/*      $Header: /ccm/meitca/hsl/zonesjagents/skeleton/StringResources.java 3      11/11/96 5:16p Billp $
 *
 *  Copyright  1996 Horizon Systems Laboratory,
 *  Mitsubishi Electric Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  The String Resources for the skeleton package
 *
 *      $Log: /com/meitca/hsl/zonesjagents/skeleton/StringResources.java $
 *
 * 3     11/11/96 5:16p Billp
 * Made correction to company name
 *
 * 2     9/12/96 11:00a Walsh
 * Automatically compile the skeleton file
 *
 * 1     9/06/96 4:54p Walsh
 *
 */
package com.meitca.hsl.zonesjagents.skeleton;

class StringResources {

        // Generator.java
        public static String USAGE = "Usage: generator agentclassname";
        public static String NO_AGENT = "generator: You must specify an agent class name.";
        public static String TOO_MANY = "generator: Too many parameters specified.";
        public static String NOT_FOUND = "generator: Agent class not found -- ";
        public static String IO_ERROR = "generator: IO error while attempting to write the skeleton file -- ";
        public static String COMPILE_ERROR = "generator: Anb error while attempting to compile the skeleton file --

}
```

WO 98/21662 PCT/US97/20232

Mitsubishi Electric ITA

Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents

November 12, 1996

Horizon Systems Laboratory
Mitsubishi Electric ITA
1432 Main Street
Waltham, MA 02154

Mitsubishi Electric ITA Confidential

Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents

1. Introduction

This document serves as a patent disclosure on the invention entitled *Dynamic Synchronous Collaboration Framework for Mobile Agents*. This document includes a discussion of: prior art, problems with prior art, details of the invention, and improvements of this invention over prior art.

2. Prior Art

"Agent" is one of the most overused terms in the software industry today. Although the term may be used in a wide variety of contexts, software Agents can be classified into two major categories: Intelligent Agents and Mobile Agents. Although the two differ in scope and capabilities, both types of Agents are able to perform work asynchronously and autonomously on behalf of their owners. Intelligent Agents are endowed with some degree of reasoning and learned behavior and may make decisions on behalf of their owners. These Agents are generally stationary and execute on a single system. Mobile Agents, on the other hand, may be transported across nodes in a network, performing computations as they migrate.

The original concept for Agents stems from the Artificial Intelligence community. The behavior of Intelligent Agents may be determined by programming or by decisions made by the Agents on behalf of their owners. As an example, an Intelligent Agent may search a database for unusual outdoor scenery, making its own decisions as to whether or not each scene meets the specified criteria.

Intelligent Agents generally employ both synchronous and asynchronous modes of communication. Agent communication may take the form of queries or collaborative computations. In the realm of Agent computing collaboration may be defined as "the coordination of the computations of multiple Agents to achieve a desired result" (e.g., to solve a complex problem). Collaboration often results from discovery (i.e. the need to

collaborate with a specific Agent regarding a certain subject is not known a priori, but is, rather, determined as a result of the Agent's computations.)

Both the interaction between these Agents and the context in which they execute must be formally specified. Standardization has not yet been achieved for these objectives. However, the prevalent means for accomplishing these results is via implementing a user model and specifying queries and assertions by using a knowledge representation language, the most popular of which is currently Knowledge Query Manipulation Language (KQML), and by developing Ontologies, or vocabularies of common terms to be utilized by interacting Agents. KQML is comprised of both a language, with a syntax unlike procedural or object-oriented languages, and a communications protocol. An Ontology is defined as "an explicit specification of a conceptualization". The syntax and semantics of an Ontology are generally extensions of a first-order predicate calculus, typically the Knowledge Interface Format (KIF).

The combination of KQML, KIF, and Ontology development is a complex task. Some implementations utilizing these mechanisms are being developed in the Artificial Intelligence community. But this approach has not caught on in other software applications.

In contrast to Intelligent Agents, Mobile Agents are usually explicitly programmed to accomplish a specific task. As an example, a Mobile Agent could migrate to travel agencies on the World Wide Web and determine the lowest cost air fare between two cities. In this case, there is no need for learned behavior or complex reasoning.

However, the amount of information accessible on distributed systems, including databases, the Internet and Intranets, is growing at a staggering pace and much of it is available in incompatible storage formats. The need to filter information and make decisions based on its content is increasing, and with it, the need for increased collaboration among Mobile Agents.

The need for collaboration is illustrated by extending the above travel agency example. A Mobile Agent may need to clone itself and instruct each of its counterparts to migrate to a different travel agency site to obtain the best package for a trip to one or more ski resorts meeting certain parameters, including air fare, lodging, meals, lift tickets, and car rental. Furthermore, the information may, itself, be stored in different formats (databases, etc.), necessitating some amount of filtering and data reformatting. Once the Agents have obtained their information, they need to collaborate with each other to correlate their results and determine the best package available at each resort for different travel dates and the overall best deal.

Mobile Agents are evolving to include increased intelligence and collaboration. Some research organizations have begun incorporating KQML and Ontologies into Mobile

Agent implementations, while other utilize different knowledge representation languages to implement Agent communication. Still, others use proprietary object languages.

3. Problems with Prior Art

The KQML/Ontology mechanism and other knowledge representation languages are ill-suited for general-purpose software applications. They require programmers to learn multiple languages to formally specify communications and to develop complex dictionaries of terms – a task that is currently ill-defined. The knowledge representation languages are also extremely unlike the procedural and object-oriented languages utilized in building business and general-purpose software applications.

The need to support asynchronous collaboration adds significant complexity to Agent systems implemented via knowledge representation languages (e.g. KQML) and Ontologies. Furthermore, there exists a class of Mobile Agents which solely require synchronous collaboration, e.g., the travel agency application described above. This type of Agent application essentially subdivides a larger task into smaller pieces and creates Agents to migrate throughout the network to accomplish it. For these Agents, there is no benefit in the added complexity of coding in multiple languages and developing Ontologies, nor in the support for asynchronous communication. A synchronous collaboration framework in the Agent's language is the best means for these Agents to coordinate their results and implement adaptive behavior.

4. Some Details of the Invention

The dynamic synchronous collaboration framework utilizes a distributed synchronization point plus object-oriented abstractions to provide synchronous collaboration for Mobile Agents in a distributed system. The framework simplifies collaboration for applications which subdivide a complex problem into several pieces and farm out the work to multiple Mobile Agents.

This framework enables Mobile Agents within an application to perform synchronous collaboration with one or more affiliated Agents in the application's native language. This is accomplished via the *Agent Group* abstraction. Agents wishing to collaborate with each other must belong to one (or more) of the same Agent Groups. The fundamental principle behind Agent Group collaboration is that Mobile Agents performing sub-computations, possibly on different nodes in a distributed system, will need to periodically correlate their results, and potentially adapt their behavior based on the results of the correlation. The Agent Group acts as a distributed synchronization point for this type of synchronous collaboration.

The dynamic nature of the Agent Group enables Agents to join and leave the group at will. Hence no a priori knowledge of the actual Agents comprising a group is required and group membership may change over time.

A distributed name space maintains a reference to each Agent Group. Hence, Agent Groups may be located via their entry the name space. When an Agent Group is created, it is automatically added to the name space.

The Agent Group's primary responsibility is to facilitate synchronous collaboration for Mobile Agents. However, it also serves as a central point for distributing asynchronous events to Agents in the group. A typical event may be the termination of a particular Agent. As a side effect, the Agent Group also tracks the travels of its constituent Agents throughout the network.

The travel agency example illustrates the function of Agent Groups. The problem to be solved can be stated as follows: A user wishes to determine the best package for a trip to a ski resort. The Agents must determine the best travel date and time (e.g. any week in February starting on a Saturday, and flying before noon) and lodging must meet certain criteria (e.g., a two-bedroom condominium). The trip includes all expenses (e.g., air fare, lodging, meals, transportation, and lift tickets). The user may also specify several possible travel destinations.

This application is composed of multiple Agents -- each is responsible for querying one or more data sources (in perhaps different formats) for specific information (e.g., searching a particular travel agency's database of promotional trips to determine if any meet the criteria specified by the user.) Some Agents may query legacy database systems, whereas others may search ODBC compliant databases. (The actual mechanisms for handling different data sources is beyond the scope of this invention.)

Before the Agents are transported throughout the network they join a new Agent Group. The Agent Group tracks its Agents' travels throughout the network. At any point, an Agent may be in one of several states. Figure 1 depicts three Agents in an Agent Group performing computations and one Agent in-transit between nodes.

At different points during their migration, Agents may wish to correlate their results, and potentially change their migration plans or behavior. Suppose each Agent migrates to a local travel agency. Before migrating to travel agencies in different cities, the Agents may wish to correlate their results and potentially adjust their behavior. Suppose one of the Agents determines that a particular ski resort, say Taos, has no two-bedroom condominiums available during February. The Agents in the group may wish to drop all further queries about trips to Taos.

Agent Group collaboration is implemented via a distributed synchronization point (also known as a collaboration point) and a software routine to analyze the results of each

Agent's computation. The Agent Group abstraction provides the distributed synchronization point. Each application need only provide its own software method to analyze the results and potentially allow each Agent to adapt its behavior.

The distributed synchronization scheme requires that each Agent arrive at the collaboration point before collaboration may commence. Hence, it is best suited to applications that subdivide a complex problem into many sub-tasks that need to correlate their results. As each Agent arrives at the collaboration point, it posts the results of its computations to the Agent Group, and blocks until all the other Agents in the group arrive. Figure 2 depicts one Agent in the group waiting to collaborate while three others are still performing computations.

The Agent Group collects the results of each Agent's sub-computation. When all Agents have arrived at the collaboration point, as shown in Figure 3, the Agent Group notifies them that collaboration may commence. When it unblocks the waiting Agents, the Agent Group passes the collected results to each Agent. Each Agent then calls the application-specific method to analyze the results and potentially adapt its behavior. Figure 4 shows that all Agents have arrived at the collaboration point and are now in the analysis stage.

The synchronous collaboration framework supports both parallel and serialized collaboration analysis. Figure 4 depicts a parallel analysis implementation; Figure 5 illustrates a serialized collaboration analysis, in which one Agent is in the Analysis state, while the rest remain waiting in the Collaborate state.

The distributed collaboration point may be implemented via an object transport or by an RPC mechanism. It may also be implemented in any procedural or object-oriented language. Object-oriented languages simplify the implementation, as the application need only subclass the Agent Group abstraction and provide a concrete method to analyze the results and potentially adapt the Agent's future behavior.

The Agent Group ensures that all Agents in the group arrive at the synchronization point by tracking the Agent's migration through the network. In the current implementation, as part of a mobile Agent system, a server (known as the Conduit Server) on each node manages the Agent's migration and informs each of its Agent Groups when the Agent arrives and departs. Should an Agent fail to arrive at a collaboration point in a pre-determined, configurable length of time, the Agent Group will "ping" the Agent by sending a message to ensure that it is still alive, as shown in Figure 6. (In the current implementation, this is performed via the Conduit Server on the node where the Agent is executing.) If the Agent fails to respond to the ping, the Agent Group notifies the other members of the group by distributing a Collaboration Failed asynchronous event to them that indicates that a particular Agent is not responding. The Agents may then indicate whether or not they wish to proceed with the collaboration. Figure 7 depicts the Agent Group's distribution of a Collaboration Failed event followed by the remaining Agents

indicating that they wish to continue the collaboration. Alternatively, the Agents could decide to abort the collaboration.

The Agent Group also implements deadlock detection by means of the time-out mechanism and Agent ping described above. The ping actually returns the state of the Agent in question (via the Conduit Server). If the Agent is already in the Collaborate state but has not arrived at this collaboration point, a deadlock has occurred. This is shown in Figure 8. In this case, the Agent Group will abort the collaboration and notify the Agents in the group of the deadlock by sending them a Deadlock Event, as shown in Figure 9. Note that since Agent Group collaboration is designed for closely coordinated Agents, the existence of a deadlock is generally a result of a programming error. Hence, the Agent Group does not need to employ a more sophisticated scheme for deadlock detection or avoidance.

Agent Groups provide reliability by saving the state of each Agent in the group to persistent storage. If the Agent Group should terminate unexpectedly, it is restarted and reads the state of the group members from persistent storage. The Agent Group's restart is transparent to the Agents. They actually communicate with an Agent Group Proxy (instead of an Agent Group), which shields them from Agent Group failures.

Figure 11 illustrates the flow of a typical collaborating Mobile Agent. Each Agent creates an Agent Group Proxy for each Agent group it wishes to join. The proxy's primary responsibility is to maintain an active connection to its associated Agent Group and to re-establish the connection or re-create the Agent Group, as needed, during Agent Group failures.

When the Agent Group Proxy is initialized, it establishes a connection with the appropriate Agent Group. It first attempts to find a reference to it in the name space. If this fails, it creates the Agent Group and enters a reference to it into the name space. The algorithm guarantees the atomicity of this operation.

After the proxy has created the Agent Group or established a connection to it, the Agent continues its computations and migrations. Eventually, it may attempt to collaborate its results. This is accomplished by means of the Agent Group Proxy. If the Agent Group has failed or is in a unknown state, the proxy will be unable to contact it. If this occurs, the Agent Group Proxy retries the communication. If, at this point, the proxy is still unable to communicate with the Agent Group, it recreates it and updates the name space with a reference to the new Agent Group. The re-creation of the Agent Group is serialized to prevent multiple Agent Group Proxies from simultaneously doing so. If a proxy determines that the Agent Group is in the process of being re-created, it simply waits until it can find a reference to the new group in the name space. Once the Agent Group has been re-created, the proxy retries the collaboration.

As mentioned earlier, Agent Groups provide a distributed events mechanism in addition to the synchronous collaboration framework. The distributed events mechanism enables the Agent Group to notify members of the group of outstanding events, such as the termination of an Agent, or the detection of a collaboration deadlock. Agents may build on this mechanism to forward application-generated events to other members in the group. Hence, if an Agent catches an application-generated exception it may notify the other Agents in the group before terminating. Figure 10 illustrates how the Agent Group forwards an Exception Event, generated by one of its Agents, to the rest to the group.

Currently, the Agent Group handles distributed events by maintaining a (possibly remote) reference to an event queue associated with each Agent. The Agent Group forwards all events it receives to the other members of the group by enqueuing it. The Agent Group may be implemented without a remote referencing capability, but this would add significant complexity. In the current implementation, each Agent contains an events thread which is responsible for dequeuing the event and calling the application-specific event handler. (This thread could easily be a process, if threads were not available.)

In addition to forwarding events to members of the group, the Agent Group may generate its own events and distribute those to the group members. As an example, the Agent Group distributes a Deadlock Event to the group members when its detects a deadlock situation, as illustrated in Figure 9.

As mentioned earlier, Agent Groups track the migrations of their constituent Agents, with help from the Conduit Server (in the current implementation). If an Agent is in transit when the Agent Group is forwarding events, it queues the events locally, as shown in Figure 10. Once the Agent arrives at its destination, it notifies the Agent Group, via the Conduit Server. The Agent Group then flushes the events it stored locally for that Agent.

Agent Groups implement varying levels of persistence, as defined by the actual configuration. Memberships and Agent states are updated infrequently, so they are generally written to persistent storage. However, local queues maintained on behalf of in-transit Agents may also be saved in persistent storage, if desired.

Implementation of the Agent Group in an object-oriented language enables applications to extend its functionality, if required, by subclassing the Agent Group object.

5. Benefits of the Invention

The work described above provides several benefits. It greatly simplifies the implementation of synchronous collaborating Mobile Agents, especially for applications that subdivide a complex task into a series of related sub-tasks. There is no need for application developers to learn knowledge query representation languages, such as KQML, or to develop complex Ontologies. The dynamic synchronous collaboration framework provides a natural programmatic implementation that may be programmed in a

number of languages. An added benefit of this scheme is that collaboration is implemented as part of the application, programmed in the same language as the rest of the application. The current implementation in an object-oriented language (Java) also enables the technology to be easily extended so that more complex intelligence schemes may be supported.

Agent Groups facilitate Mobile Agent tracking and may also be queried by tools that monitor Agents. In addition, collaboration may be performed through discovery, as with many knowledge representation languages.

Agent Groups also offer reliability by providing persistent Agent Group membership and Agent state information. Furthermore, Agent Group Proxies shield Agents from the effects of Agent Group failures. They detect an Agent Group's failure and automatically restart the failed Agent Group (which then restores the state of its membership from persistent storage) and retry the current operation (e.g., collaboration). The algorithm also guarantees the atomicity of Agent Group creation and re-creation.

An Agent Group also detects Agent failures and deadlocks and forwards this information to the other members of the group via its distributed events mechanism. Furthermore, this mechanism allows Mobile Agents to asynchronously notify associated Agents of relevant events, such as the catching of an exception.

Implementation of Agent Groups in an object-oriented language also enables additional application-specific functionality to be easily added via inheritance and polymorphism.

6. Figures

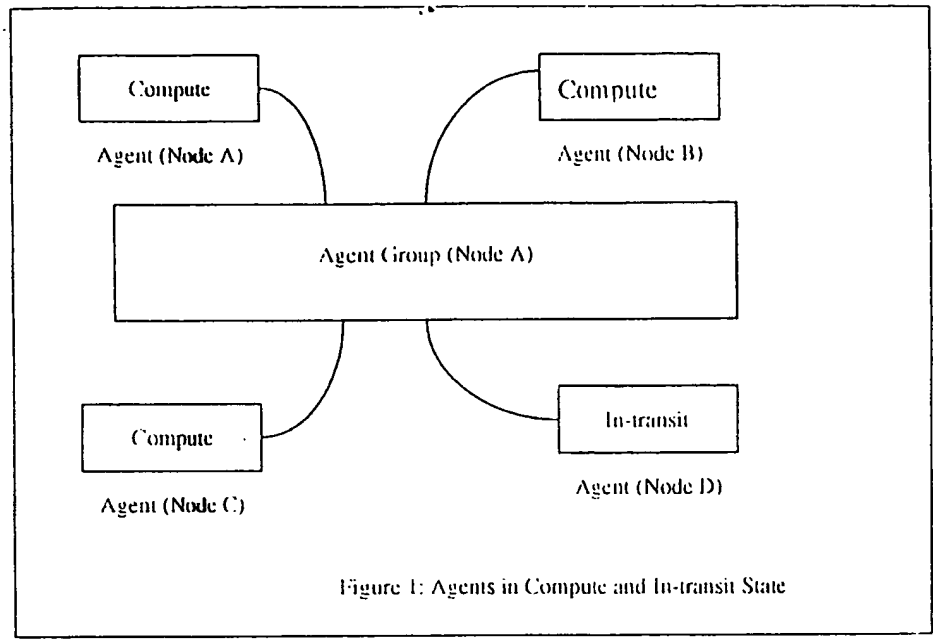


Figure 1: Agents in Compute and In-transit State

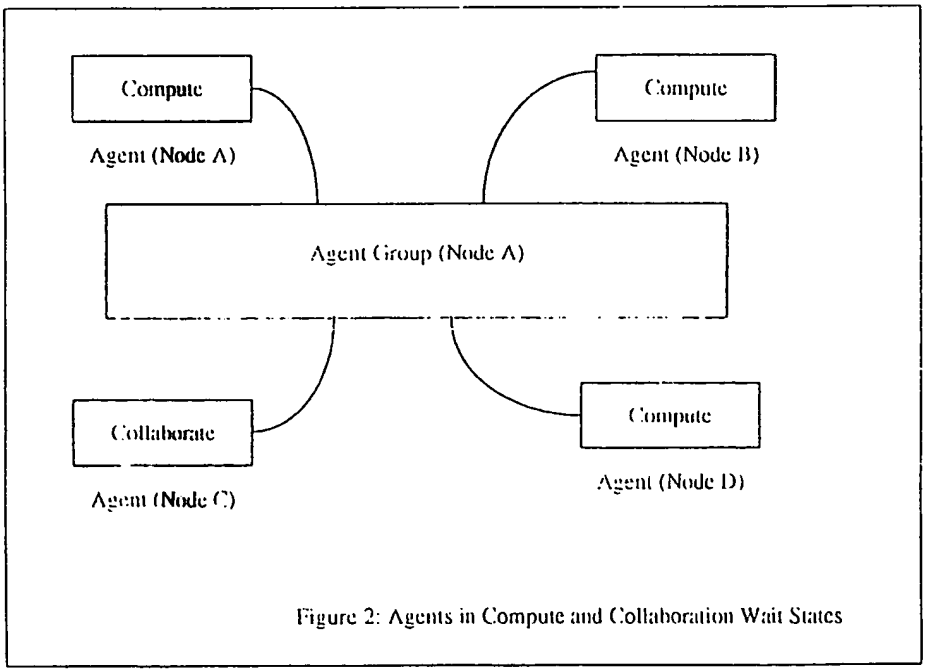


Figure 2: Agents in Compute and Collaboration Wait States

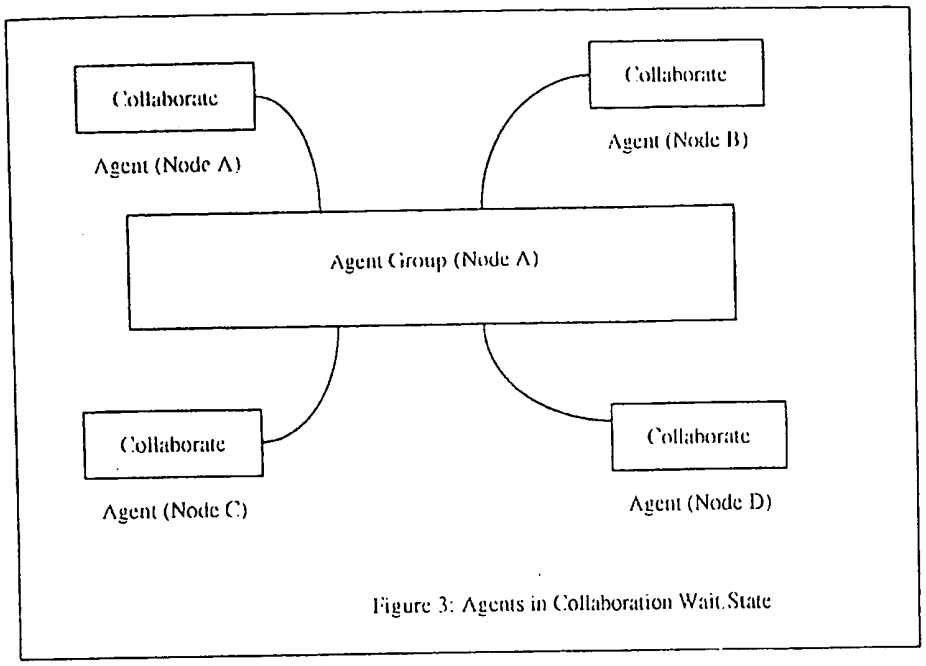


Figure 3: Agents in Collaboration Wait.State

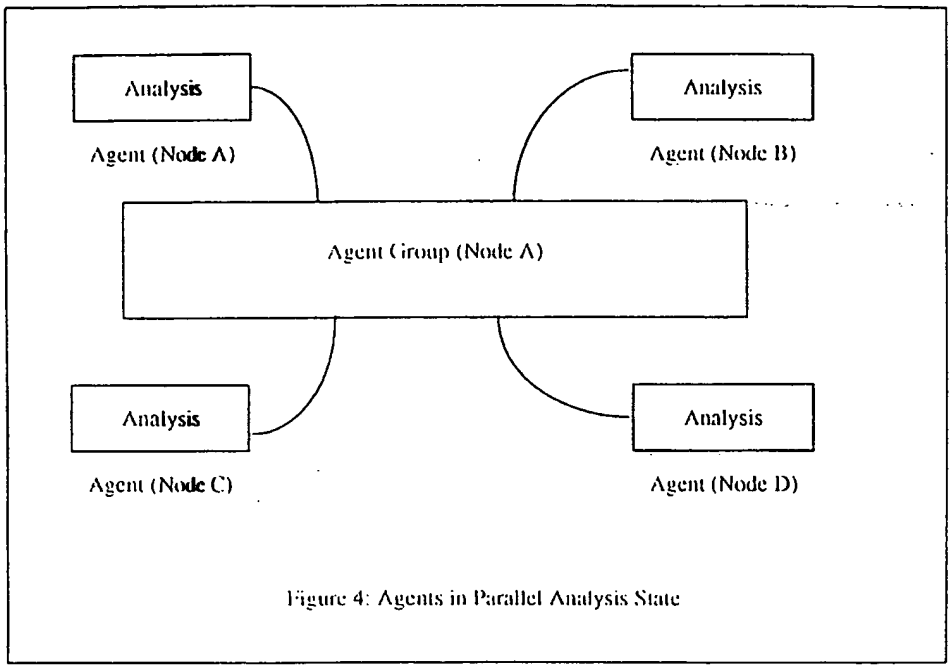


Figure 4: Agents in Parallel Analysis State

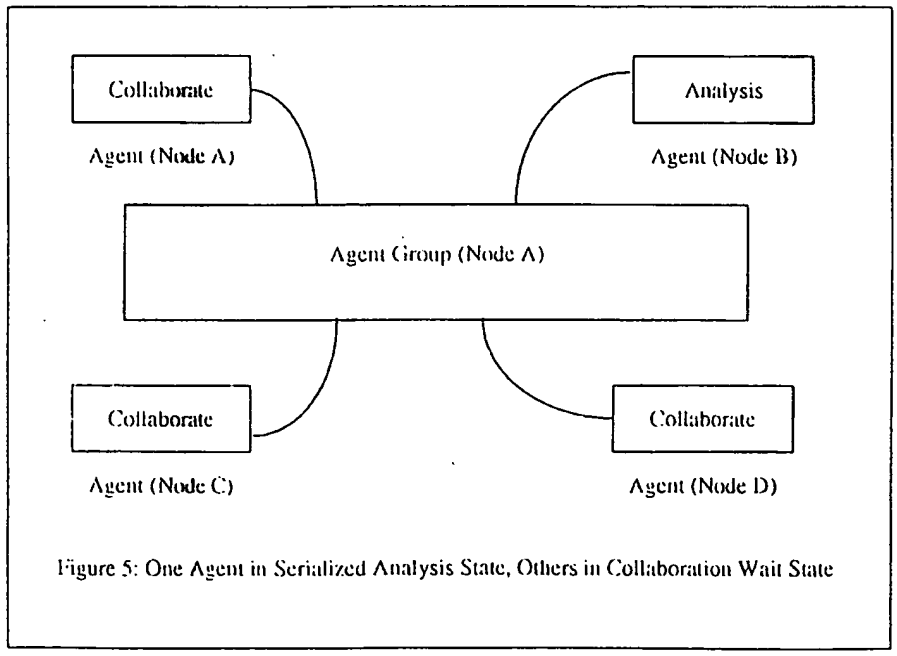


Figure 5: One Agent in Serialized Analysis State, Others in Collaboration Wait State

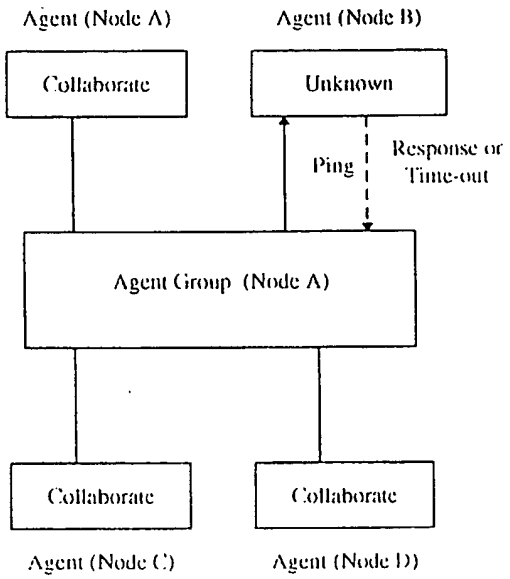


Figure 6: Agent Group Pings Agent After Collaboration Times Out

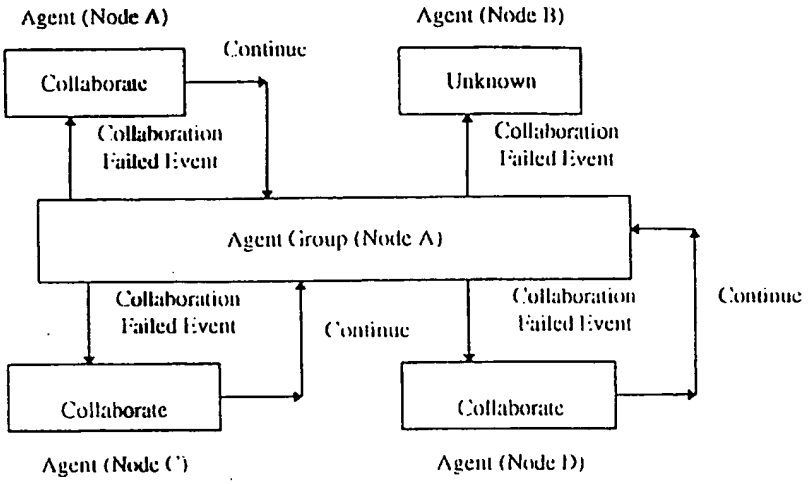


Figure 7: Continued Collaboration Despite the Failure of An Agent to Respond

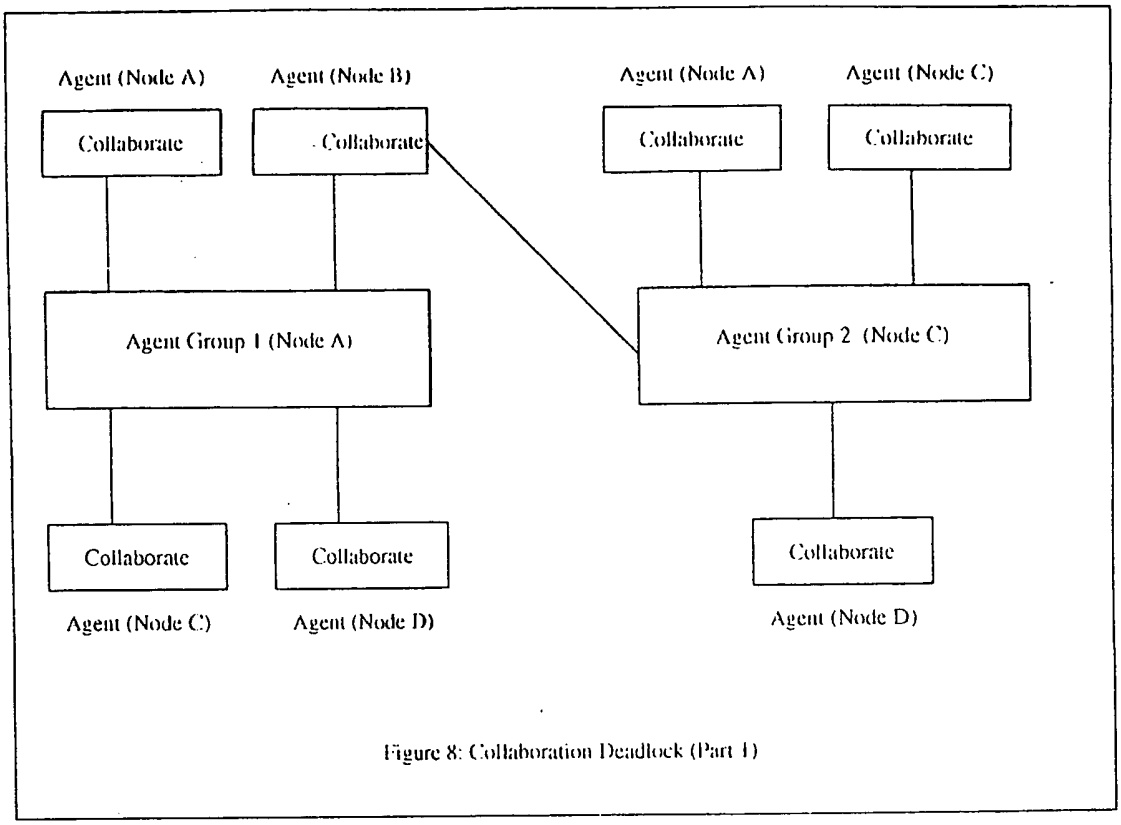


Figure 8: Collaboration Deadlock (Part 1)

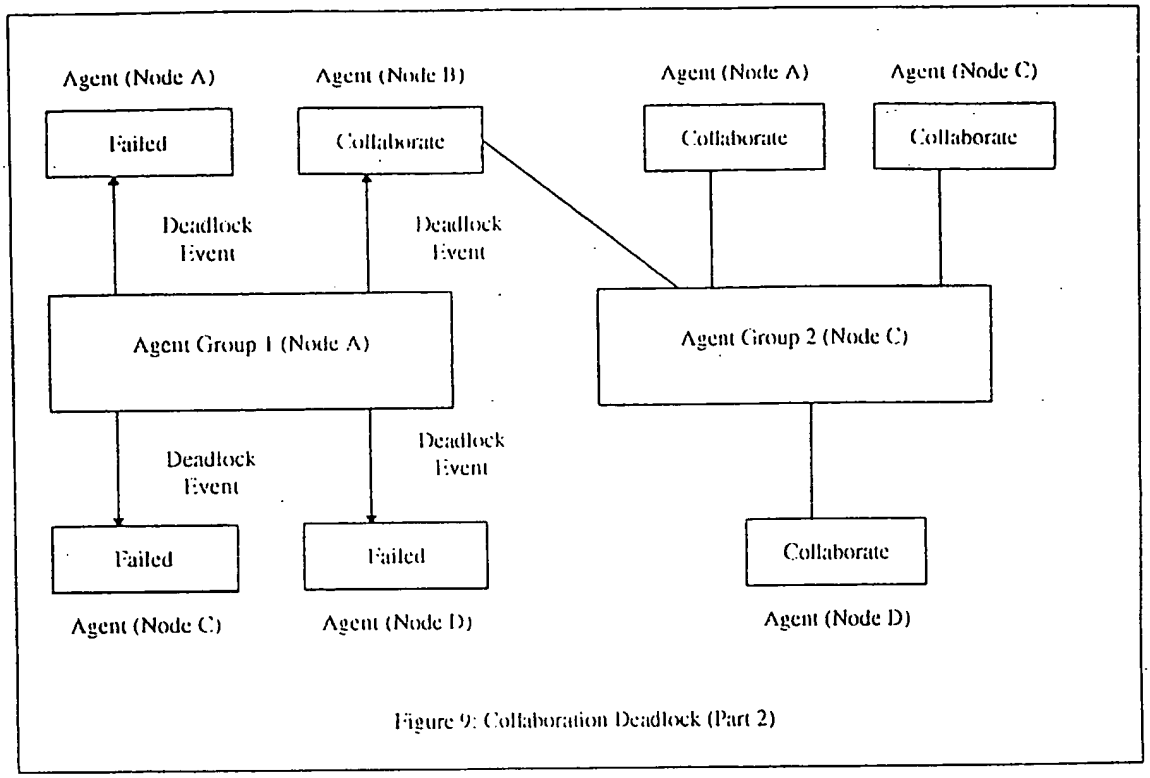


Figure 9: Collaboration Deadlock (Part 2)

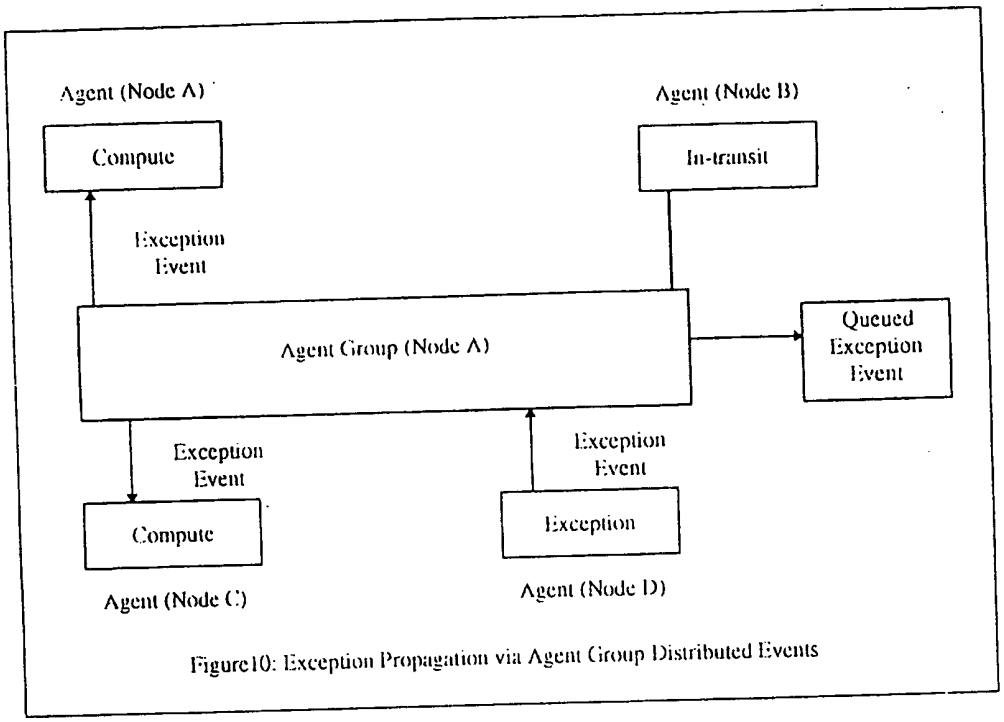


Figure10: Exception Propagation via Agent Group Distributed Events

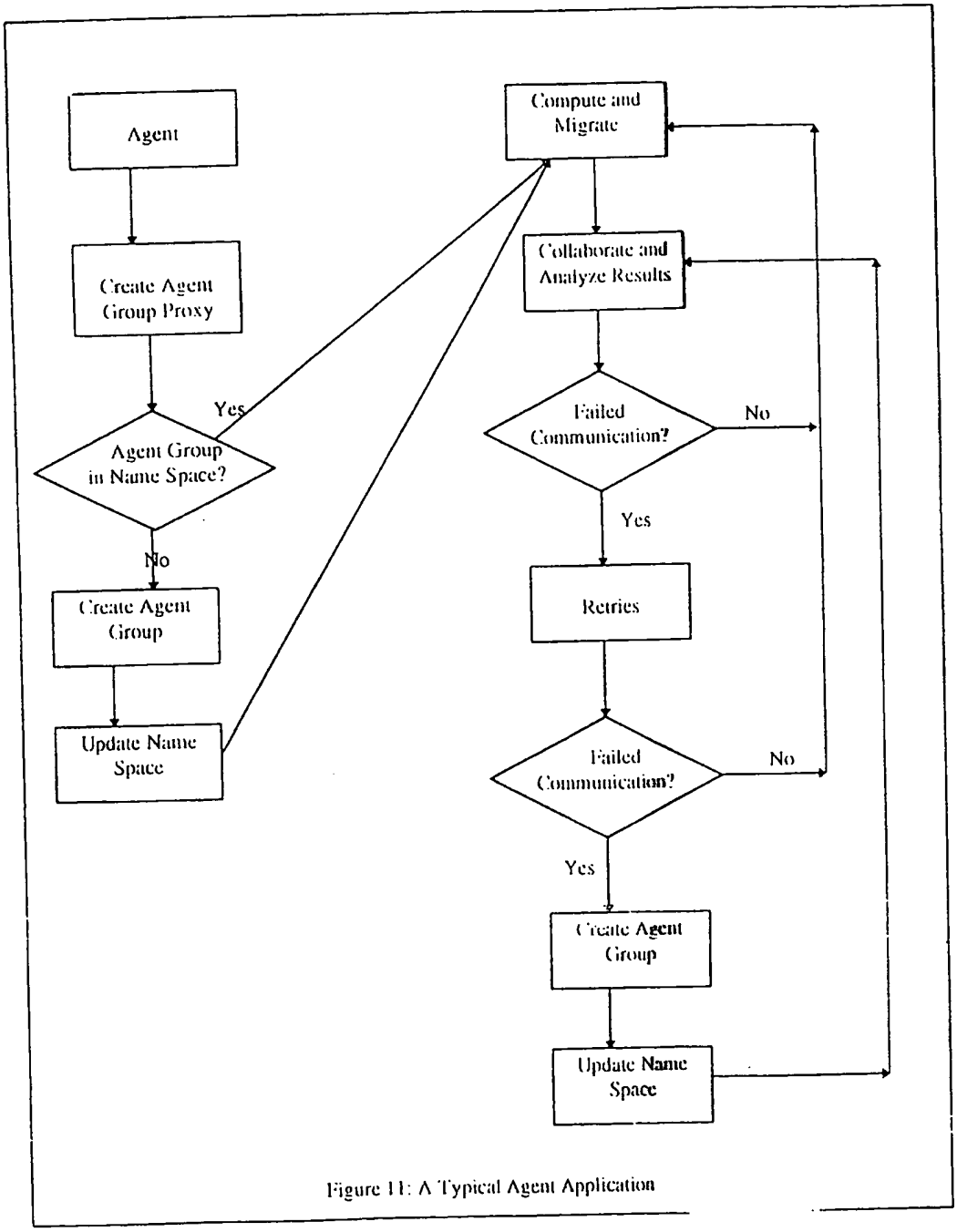


Figure 11: A Typical Agent Application

Mitsubishi Electric ITA

Design Specification for Java Agent Coordination

Version 0.2
November 12, 1996

Horizon Systems Laboratory
Mitsubishi Electric ITA
1432 Main Street
Waltham, MA 02154

Mitsubishi Electric ITA Confidential

Mitsubishi Electric ITA Confidential

Design Specification for Java Agent Coordination

Version 0.2[1]

1. Introduction

Java agents have the capability of coordinating their activities. This coordination takes two main forms: synchronized collaboration and asynchronous notifications. These modes of communication complement each other and their implementations are related.

Collaboration

The collaboration paradigm is implemented as follows:

- multiple agents perform computations at different nodes in the network;
- after performing all or a subset of their operations, agents wait at a distributed synchronization point until all agents have "arrived";
- agents coordinate by potentially serialized execution of a synchronized method.

Coordination can take the form of performing computations on intermediate results and/or updating agent itineraries.

Notification

The notification scheme uses a distributed events mechanism to communicate agent state changes that occur asynchronously. It is mainly utilized for distributed exception handling, but may also be used to notify related agents of events, such as another's termination.

As described above, agent coordination utilizes distributed synchronization and event handling. However, Java possesses neither of these. Actually, its synchronization primitives are limited to: critical sections on a single system and a notification scheme that can only handle a single type of event. Hence, a great deal of the complexity in the task of agent coordination is comprised of implementing mechanisms to overcome Java's unsuitability for distributed processing. These mechanisms are described in the remainder of this document, along with the new objects that implement them

[1] Revision History:
07/26/96 Version 0.1 Initial revision.
11/12/96 Version 0.2 Major changes to Notifications and Exceptions and Collaboration sections to reflect actual implementation. Removed Events section.

2. Synchronization

As mentioned above, agent coordination requires distributed synchronization mechanisms, such as locks and events. Since Java does not provide any distributed locking, we are faced with the following choice:

1. Utilize a centralized locking scheme.
2. Develop our own distributed locking paradigms.
3. Use Java's native methods to interface with the operating system's synchronization primitives.

The latter choice is unacceptable because it compromises the portability of our agent infrastructure, whereas the first option may potentially introduce considerable network overhead. The second alternative also requires significant development effort even to implement a well-known solution to the problem.

In the interest of expediency (as required by the tightness of our schedule), the initial Java agent infrastructure utilizes RMI to implement a centralized locking scheme. Locking is implemented via method invocations on a potentially remote object (called an *AgentGroup*. See Section 3 for details.) Agent coordination is implemented by (remote) invocations on a method which requires synchronized access. (See Section 3.)

3. CollaboratorAgent and AgentGroup

Agents that may wish to collaborate their results subclass *CollaboratorAgent*, an abstract class derived from *Agent*. CollaboratorAgent contains a minimum of two threads – one to perform computation and to travel and another to handle asynchronous event notifications. Applications that contain collaborating agents subclass CollaboratorAgent and, of course, these derived classes may implement several computation threads.

Each collaborating agent belongs to one or more agent groups, which are described by Agent Group objects. Agent Group is actually a distributed RMI object. *AgentGroup* defines the interface; *AgentGroupImpl*, an abstract base class derived from *PersistentEventGroupImpl*, provides the implementation. (See *Design Specification for Distributed Java Events* for a description of PersistentEventGroupImpl.)

An AgentGroup is essentially a unit of coordination. Hence agents may coordinate with all the other agents in an AgentGroup. They may also create new AgentGroups on demand, and join and leave groups at will. (In practice, AgentGroup must be subclassed to implement agent collaboration. However, this document does not distinguish between the base class and its derived classes and refers to them collectively as AgentGroups.)

An AgentGroup is an object that is essentially shared among the members of an agent coordination group. It is owned by the Agent that initially creates the group (an agent group is created by instantiating AgentGroup) and accessed via a potentially remote RMI reference.

The relationship between CollaboratorAgent and AgentGroup is similar to that of Thread and ThreadGroup: A ThreadGroup maintains references to the Threads composing the group; an AgentGroup maintains a similar list of its CollaboratorAgents. Both ThreadGroups and AgentGroups form a hierarchy and each contains a reference to its parent group. Thread's constructors are either passed a reference to a ThreadGroup or they save a

reference to the parent's Thread's group. Likewise, CollaboratorAgent's constructors may be passed one or more AgentGroups or save a reference to AgentGroup of its parent, if it exists.

However, there is one noticeable difference between the two types of groups. A Thread may only belong to one ThreadGroup at a time, whereas a CollaboratorAgent may simultaneously be a member of several AgentGroups, which constitute a hierarchy. Hence, Threads contain a reference to their associated ThreadGroup, but CollaboratorAgents maintain a list of references to their AgentGroups.

Support for multiple AgentGroups facilitates coordination among related groups of agents and also enables the creation of an AgentGroup hierarchy through which communication may be propagated. The pilot application may utilize multiple agent groups, as shown in Figure 1.

This figure portrays an example of two AgentGroups: AgentGroup A consists of three agents – one on each LAN in the system, whereas AgentGroup B contains agents on multiple nodes on LAN 3. A distributed application running on a configuration such as this typically creates multiple AgentGroups when it needs to traverse the WAN. In this case, 3 agents are created in AgentGroup A, one for each LAN. Once they arrive on a node, these agents clone themselves, creating more agents to be distributed across nodes on the LAN (e.g., AgentGroup B). This approach minimizes network traffic over the WAN and forms a multi-level hierarchy of AgentGroups. Agents in the intra-LAN groups (e.g., AgentGroup B) can communicate their results, which can then be coordinated with the inter-LAN group (i.e., AgentGroup A).

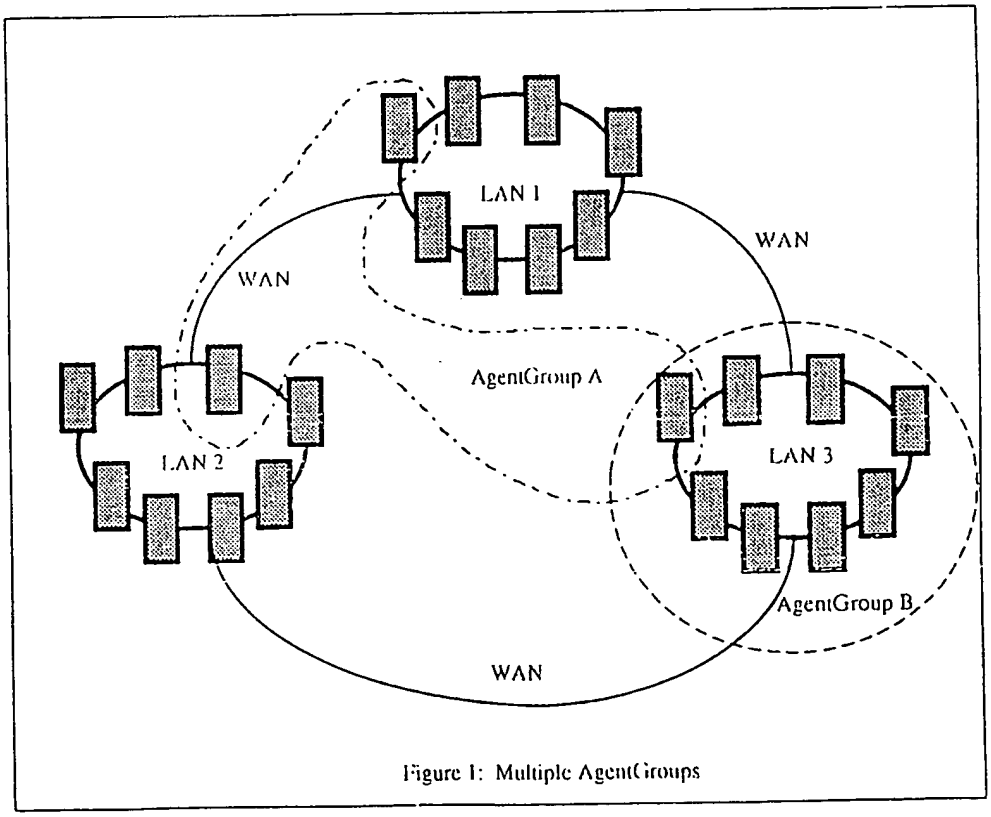


Figure 1: Multiple AgentGroups

By default, RMI passes all local objects by copy. However, the above scheme requires that each Agent actually maintain (remote) references to its AgentGroups. Hence, the agent infrastructure overrides some of RMI's object serialization methods to generate the required references.

4. Notifications and Exceptions

Both asynchronous notification and collaboration are implemented via the associated AgentGroup object. The following description serves as an example of asynchronous notification handling via distributed events. (See *Design Specification for Distributed Java Events* for more details on event handling.)

1. An agent catches an exception and wishes to notify the other members of the group.
2. The agent posts an *AgentExceptionEvent* to the AgentGroup via the group's *postEvent* method. The AgentExceptionEvent indicates the type of exception that occurred.
3. The AgentGroup posts the AgentExceptionEvent to the other agents in the group.
4. Each agent's Event Handler performs application-specific handling of the ExceptionEvent.

If the agent that caught the exception wishes to notify another associated Agent Group, it repeats the above procedure with the next AgentGroup object.

Occasionally, the Agent Group may generate events that it directly forwards to a member of the group (e.g., to "ping" it after it fails to arrive at a collaboration point) or others that it "multicasts" to its members (e.g., an Agent's failure to reply to a ping).

5. Collaboration

Agent collaboration is more complicated than asynchronous notifications. It is implemented as follows:

1. An agent performs some computation and arrives at a synchronization point.
2. The agent performs a (remote) invocation of the AgentGroup's *collaborate* method, passing it an *AgentStatus* object.
3. If no collaboration is in-progress (i.e., this is the first agent that has requested collaboration), collaborate calls its synchronized *beginCollaborate* method, which initializes the number of agents to wait for before collaboration may begin (i.e., n-1 for an n-agent group.) (Race conditions are handled by checking if the agent count has been initialized (i.e., it is non-zero) before continuing. A race implies that two agents are attempting to initiate the same collaboration. Otherwise the application is semantically incorrect.) This routine saves the requesting agent's unique ID (obtained from the AgentStatus object) and starts a collaboration timer (with sufficient time for agents to arrive across the WAN) by passing a configurable time-out value to the AgentGroup's *wait* method.
4. Otherwise (i.e., if a collaboration is in-progress), collaborate calls the synchronized *decrementCollaborate* method to add the requesting agent's unique ID to the list of

agents arriving at the collaboration point and to decrement the waiting agent counter. If the counter is non-zero, this method blocks with no time-out value. Otherwise, the Agent Group awakens the other threads waiting for collaboration to commence.

5. If the timer expires, all agents did not arrive at the collaboration point in time. This may occur because an agent received an uncaught exception and terminated unexpectedly or because a deadlock condition exists. (Since agents may belong to multiple groups, overlapping collaborations could result in deadlock. However, a properly coded application will not generate deadlocks because the collaboration points must be known a priori and must be executed in the same order among collaborating agents, i.e., collaboration must first handled at the leaf Agent Group and propagated up the hierarchy.) If a time-out occurs, the Agent Group thread initiating the collaboration (on behalf of its associated CollaboratorAgent) is awakened and performs the following:

   - It calls the AgentGroup's *checkCollaborate* method which returns the collaboration state (either *CollaborationSucceeded* or *CollaborationFailed*, depending on the value of the waiting agent count).

   - If the collaboration failed, the collaborate method calls *recoverCollaborate*. This method attempts to communicate with the agents that did not arrive at the collaboration point by "pinging" them. (It determines these agents by comparing the list of arrived agents with the group membership.) The ping is actually performed via a message to the Conduit Server on the node where each agent was last known to be executing. If the agents are located and any of them is in the *Collaborate* state and but has not arrived at this collaboration point, a collaboration deadlock has occurred and the collaboration state is set to *CollaborationDeadlock*. If any of the agents cannot be located, the collaboration state is set to CollaborationFailed. Otherwise, it is set to *CollaborationRetry* and the Agent Group thread retries the collaboration up to a configurable number of times (after which the state will be set to CollaborationFailed if the collaboration still times out.)

   - The initiating Agent Group thread wakes up the other blocked threads by calling *notifyAll* on the AgentGroup.

6. Once all the collaborating Agent Group threads have been awakened, (either because the counter has dropped to 0 or a time-out occurred), they call the AgentGroup's *checkCollaborate* method. If this method returns a state other the CollaborationSucceeded, each thread returns the appropriate exception (i.e., either *CollaborationFailedException* or *CollaborationDeadlockException*) to its invoking CollaboratorAgent.

7. If the collaboration state is CollaborationSucceeded, collaboration commences by executing a potentially synchronized application-specific method, *analyzeResults*, to perform the collaboration. analyzeResults is an abstract AgentGroup method that must be implemented by derived classes to perform application-specific collaboration. As a result of collaboration, agent itineraries may be updated.

8. As each thread completes its collaboration, it replies to the remote *collaborate* invocation, passing any state specified by the implementation, (e.g., updated itineraries).

9. The agents then continue execution on their current node and travel as required by their itineraries.

As described above, collaboration requires complex mechanisms. However, most of these are necessitated by the limited synchronization primitives provided by Java.

Mitsubishi Electric ITA

Design Specification for Distributed Java Events

Version 0.2
November 12, 1996

Horizon Systems Laboratory
Mitsubishi Electric ITA
1432 Main Street
Waltham, MA 02154

Mitsubishi Electric ITA Confidential

Mitsubishi Electric ITA Confidential

Design Specification for Distributed Java Events

Version 0.2[1]

1. Introduction

This document describes a general-purpose solution to distributed event management.

2. Overview

This section provides a high-level description of the objects that implement the distributed events framework.

2.1. Objects

Events

Events are described by classes that derive from *EventType*. Each EventType object includes: a unique ID (a String) generated by the constructors and a description of the event (a String) optionally passed to the constructors.

Event Handler

The *EventHandler* interface specifies the *handleEvent()* method which implements an Event Handler. Each application must provide its own Event Handler to perform application-specific event management.

Event Notification

The *EventPost* interface specifies the *postEvent()* method which implements event notification. Several of the classes described below implement this interface.

Event Queue

Any object receiving notification of events must own an Event Queue and an associated thread that manages the queue. The Event Queue's constructor is passed an EventHandler stub. It instantiates an *EventQueueThread* (a protected class derived from Thread) to manage the queue and passes the EventHandler stub to its constructor. The EventQueueThread's *run()* method removes events from the queue and calls the handleEvent() method.

---

[1] Revision History:
09/26/96 Version 0.1 Initial revision.
11/12/96 Version 0.2 Expanded discussion of Event Manager and Event Group. Added Reliability Features and Garbage Collection sections.

An Event Queue is a distributed RMI object. EventQueueImpl implements the *EventPost* interface by appending the specified EventType object to the queue and calling the associated *EventQueueThread*'s *notify*() method. This action wakes up the EventQueueThread (if it was blocked). The thread then proceeds to dequeue each pending event and pass it to the handleEvent() method

Event Manager

The Event Manager handles the registration, posting, and notification of events. When objects request notification of specific event types, they must pass the Event Manager a reference to their EventPost interface. The Event Manager saves the reference in a hash table that it uses to map event types to their notification requests. When an object posts an event (via the Event Manager's postEvent() method), the Event Manager obtains the list of objects to notify from the hash table and calls each of their postEvent() methods.

The Event Manager also maintains a list of objects that have requested notification of all events. Whenever it receives an event, the Event Manager also forwards it to these objects, by calling each of their postEvent() methods.

EventManager is an RMI distributed object. *EventManager* describes the interface, whereas *EventManagerImpl* provides the implementation. EventManager defines methods to manage event registrations (e.g., add and delete registrations).

EventManagerImpl also implements the EventPost interface. Its postEvent() method is passed an event from its originator and notifies the recipients of the event by invoking their postEvent() methods.

At most one Event Manager runs on each node in the Agent system. A reference to it can be obtained via the RMI Registry.

Event Group

A group of objects interested in a common set of notifications may form an Event Group. EventGroup is a distributed RMI object: *EventGroup* defines the interface; *EventGroupImpl* consists of the implementation. EventGroup defines methods that manage the group (e.g., add and remove members).

Whereas the Event Manager filters posted events on behalf of its clients, forwarding them only the events they requested, the Event Group performs no filtering. EventGroupImpl implements the EventPost interface. However, its postEvent() method forwards each event it receives to every member of the group (by calling their postEvent() methods).

In practice, Event Groups should be used sparingly because they could allow an abusive application to generate unwanted notifications. The Agent framework does utilize this mechanism, but it verifies that all events are generated by a group member and it only utilizes the Event Group for some well-known events, such as the termination of an Agent.

Each Event Group is identified by a unique name either passed to its constructor or assigned by it. Objects may obtain a handle to a particular Event Group by presenting the RMI registry with this unique name.

2.2. Reliability Features

The Distributed Events framework incorporates several reliability features. These generally take the form of persistent state and proxy objects.

Event Manager

The Distributed Events framework shields objects from the effects of Event Manager failures in two ways:

- by saving its state (i.e., event registrations) to persistent storage, and
- by encapsulating its functionality in a proxy object.

The Event Manager saves the following items in the persistent store:

- the hash table mapping specific event types to the objects registered to receive them and
- the list of objects to be notified of all events.

Each time an object either registers or "unregisters" its interest in receiving any (or all) events, the Event Manger writes the updated event registrations (i.e., either the hash table or the list of objects, but not both) to persistent storage via the Persistent Storage Manager. These are fairly infrequent operations, so the performance impact of this process should be negligible. (See Design *Specification for Java Agent Persistence Package* for details on the Persistent Store Manager.)

Whenever the Event Manager is started, its initialization code attempts to recover any existing event registrations from persistent storage. Hence, registrations are not lost when the Event Manager fails.

In addition, the Event Manager Proxy also prevents Objects from experiencing the effects of Event Manager failures. It acts as a wrapper object for the Event Manager. Each Object wishing to communicate with the Event Manager must first create an *EventManagerProxy* object. The proxy obtains a reference to the Event Manager via the RMI Registry. If, at any point, the proxy is unable to communicate with the Event Manager, it will attempt to obtain a new reference from the registry. It will retry this operation several times before throwing an exception.

Furthermore, if the Event Manager should fail, it will automatically be re-started by the Java Agent System.

Persistent Event Group

The Distributed Events framework provides a general-purpose implementation of Event Groups (i.e., EventGroup), as described above. However it also provides a reliable implementation of Event Groups: Persistent Event Groups. Persistent Event Group is a distributed RMI object that extends EventGroup: the interface is defined by *PersistentEventGroup* and the implementation is provided by *PeristentEventGroupImpl*, a subclass of EventGroupImpl. Persistent Event Group utilizes persistent storage in much the same manner as the Event Manager. Whenever an Object joins or leaves the group, it saves a reference to it in persistent storage.

Each Persistent Event Group is also encapsulated in a proxy object: *PersistentEventGroupProxy*. This proxy operates somewhat differently than the Event Manager Proxy, depending on how it is constructed. In some cases, it will attempt to locate a named Persistent Event Group in the RMI registry. If a name is not supplied to the constructor, it will create a new Persistent Event Group and register it.

Since Event Groups are temporary objects, unlike the Event Manager, which is a persistent server, their proxies cannot expect their associated Persistent Event Groups to be automatically re-started upon failure. Hence, a proxy may re-create the group if it is unable to obtain a reference to it after sufficient retries.

This, however, introduces a race condition wherein multiple proxy objects may simultaneously attempt to re-create the Persistent Event Group. This is handled by the following mechanism:

- If the owner of the Persistent Event Group detects the failure, it re-creates the group. (The owner is defined as the proxy that initially created the group.)
- If the owner does not detect the failure in a configurable length of time, other proxies may attempt to re-create it.
- The Persistent Event Group is re-created via the following steps:
  - Attempt to create a temporary binding in the RMI name space with a well-known name derived from the Persistent Event Group's unique name.
  - If the binding does *not* return an *AlreadyBoundException*, proceed to re-create the PersistentEventGroup and then remove the temporary binding. (Otherwise, another proxy is re-creating it. So wait for its completion.)

2.3. Garbage Collection

Both the Event Manager and Event Group objects must be prepared to garbage collect stale references. This is simply performed by deleting any objects (i.e., EventPost stubs) that it cannot communicate with. Hence their postEvent() methods handle garbage collection after notifying all interested objects of the event.

3. Changes to Agent Implementation

CollaboratorAgent

The *CollaboratorAgent* class contains a reference to an event queue (i.e., an EventQueueImpl object), which is instantiated by its constructors.

The Event Queue does not persist across an Agent's travels. Hence, its *prepareForTransit()* method flushes the queue and voids the Agent's reference to the queue before it travels. Its associated Agent Groups save any events that occur while the Agent is in-transit. The *completedTransport()* method, called after the Agent arrives at its destination, constructs a new EventQueueImpl object and flushes any events saved by the associated Agent Groups while the Agent was in-transit.

AgentStatus

The *AgentStatus* class contains a reference to the EventPost interface implemented by the CollaboratorAgent's Event Queue. It also contains an Agent's unique ID and current status. Hence, it will serve as both an Event Queue and a status indicator.

AgentGroupImpl

The *AgentGroupImpl* class derives from PersistentEventGroupImpl. Thus, it not only enables Agent collaboration, but also provides reliable group event handling.

As a result, Agent Groups may implement different levels of persistence: Group membership is always written to persistent storage, but queued events for in-transit agents will also be saved if the Agent Group's configuration allows it.

Occasionally, the Agent Group may generate events that it directly forwards to a member of the group (e.g., to "ping" it after it fails to arrive at a collaboration point) or others that it "multicasts" to its members (e.g., an Agent's failure to reply to a ping). Group members may also broadcast an event to the group (e.g., an Agent may inform the group that it has encountered an exceptional condition).

AgentEvent

The *AgentEvent* class is the superclass of all events generated by Agents. It includes the Agent's unique ID.

```
/*  $Header: /com/meitca/hsl/zonesjagents/collaborate/AgentGroup.java 5     10/01/96 3:40p Noemi $
 *
 *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
 *  Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Agent Group distributed interface.
 *
 *      $Log: /com/meitca/hsl/zonesjagents/collaborate/AgentGroup.java $
 *
 * 5     10/01/96 3:40p Noemi
 * Added some javadoc comments.
 *
 * 4     9/10/96 5:07p Noemi
 * Changed return value of collaborate method from a void to an Object.
 *
 * 3     9/06/96 3:54p Noemi
 * Added UpdateAgent method.  Changed argument to addAgent.
 *
 * 2     8/30/96 4:24p Noemi
 * Added addAgent, removeAgent, and getGroupSize methods.
 *
 * 1     8/28/96 2:13p Noemi
 * Basic AgentGroup functionality.   No support for remote groups or agent
 * mobility yet.
 */

package com.meitca.hsl.zonesjagents.collaborate;

import java.util.NoSuchElementException;

import java.rmi.Remote;
import java.rmi.RemoteException;


/**
 * AgentGroup is an RMI distributed interface.    The actual
 * implementation of AgentGroup is handled by AgentGroupImpl.
 * This class provides RMI glue that enables Agents to invoke
 * (remote) operations on an associate agent group.
 *
 * Agents may belong to one or more agent coordination groups
 * (AgentGroups), which act as focal points for distributed
 * synchronous agent collaboration and asynchronous notifications
```

```
48     * among members of the agent group.
49     *
50     * @see     Agent
51     * @see     AgentGroupImpl
52     * @author Noemi Paciorek
53     */
54
55    public interface AgentGroup
56       extends Remote {
57
58       // Instance methods
59       /**
60        * Collaboration interface.
61        * @param       result  The result of the agent's computation.
62        * @exception  RemoteException If an error occurrs setting up network
63        * connections.
64        * @exception  AgentGroupException If the collaboration times out.
65        * @return      An Object that describes the result of collaboration
66        */
67       public Object collaborate(AgentResult result)
68           throws RemoteException, AgentGroupException;
69
70       /**
71        * Adds an Agent to the group.
72        * @param   status  The AgentStatus object that describes the Agent.
73        * @exception  RemoteException If an error occurrs setting up network
74        * connections.
75        */
76       public void addAgent(AgentStatus status)
77           throws RemoteException;
78
79       /**
80        * Locates and removes the AgentStatus object containing the specified ID.
81        * @param   agentID An Agent's unique ID.
82        * @exception  RemoteException If an error occurrs setting up network
83        * connections.
84        * @exception NoSuchElementException    If the AgentStatus object does
85        * not exist.
86        */
87       public void removeAgent(String agentID)
88           throws RemoteException, ArrayIndexOutOfBoundsException, NoSuchElementException;
89
90       /**
91        * Replaces the AgentStatus object for the specified ID.
92        * This method may be used to change an Agent's status, ID, or both.
93        * @param   agentID An Agent's unique ID
94        * @param   newStatus   An AgentStatus object to replace the existing one.
```

WO 98/21662 PCT/US97/20232

```
95        * @exception  RemoteException If an error occurrs setting up network
96        * connections.
97        * @exception NoSuchElementException    If the AgentStatus object does
98        * not exist.
99        */
100      public void updateAgent(String agentID, AgentStatus status)
101          throws RemoteException, ArrayIndexOutOfBoundsException,NoSuchElementException;
102
103      /**
104       * Returns the number of Agents in the group.
105       * @return The number of Agents in the group.
106       * @exception  RemoteException If an error occurrs setting up network
107       * connections.
108       */
109      public int getGroupSize()
110          throws RemoteException;
111
112  /*
113   * TURN ON LATER
114   *
115
116      public void agentException(String agentID, Exception type) throws RemoteException;
117
118      public void agentTermination(String agentID) throws RemoteException;
119
120  */
121
122  }
123
```

WO 98/21662 PCT/US97/20232

```
/*  $Header: /com/meitca/hsl/zonesjagents/collaborate/AgentGroupException.java 2     10/01/96 3:41p Noemi $
 *
 *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
 *  Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Agent Group exceptions.
 *
 *      $Log: /com/meitca/hsl/zonesjagents/collaborate/AgentGroupException.java $
 *
 * 2     10/01/96 3:41p Noemi
 * Added a protected default constructor and some javadoc comments.
 *
 * 1     8/28/96 2:16p Noemi
 * AgentGroup related exceptions.
 */

package com.meitca.hsl.zonesjagents.collaborate;

import com.meitca.hsl.zonesjagents.shared.Agent;

/**
 * AgentGroupException defines exceptions that occur during
 * AgentGroup operations.
 *
 * @see     Agent
 * @see     AgentGroup
 * @author Noemi Paciorek
 */

public class AgentGroupException extends Exception {

  //Constructors

  /**
   * Constructs an AgentGroupException.
        * @exception   IllegalAccessException  Whenever called.
        */
  protected AgentGroupException()
      throws IllegalAccessException {

      System.out.println("Default constructor called for AgentGroupException");
      throw new IllegalAccessException("AgentGroupException Default constructor");
  }
```

WO 98/21662 PCT/US97/20232

```
48
49      /**
50       * Constructs an AgentGroupException.
51       * @param   type    A String describing the exception.
52       */
53      public AgentGroupException(String type) {
54          super(type);
55      }
56
57  }
```

```
1  /*  $Header: /com/meitca/hsl/zonesjagents/collaborate/AgentGroupImpl.java 7      10/01/96 3:43p Noemi $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
4   *  Information Technology Center America.
5   *      All rights reserved.
6   *
7              Thread.currentThread().stop();
8   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
9   *
10  *  DESCRIPTION
11  *  Agent Group distributed class.
12  *
13  *      $Log: /com/meitca/hsl/zonesjagents/collaborate/AgentGroupImpl.java $
14  *
15  * 7     10/01/96 3:43p Noemi
16  * Added javadoc comments.
17  *
18  * 6     9/20/96 12:37p Noemi
19  * Added agentsWithStatus() method and some temporary retry logic to
20  * collaboarte()
21  *
22  * 5     9/10/96 5:05p Noemi
23  * Changed collaborate and analyzeResults methods to return an Object.
24  *
25  * 4     9/06/96 3:53p Noemi
26  * Added removeAgent, updateAgent, and indexOfAgent methods.  Changed
27  * addAgent's argument to an AgentStatus object.
28  *
29  * 3     9/04/96 3:14p Noemi
30  * Added getGroupName method.
31  *
32  * 2     8/30/96 4:28p Noemi
33  * Turned on code in constructor and finalizer that registers and
34  * deregisters with the RMI name server.
35  *
36  * 1     8/28/96 2:16p Noemi
37  * Basic synchronous agent collaboration functionality.  No support for
38  * remote groups, agent mobility, or asynchronous notifications.
39  */
40
41  package com.meitca.hsl.zonesjagents.collaborate;
42
43  import java.util.*;
44  import java.io.*;
45
46  import java.rmi.server.UnicastRemoteServer;
47  import java.rmi.RemoteException;
```

WO 98/21662 PCT/US97/20232

```
48  import java.rmi.Naming;
49  import java.rmi.NotBoundException;
50
51  import java.net.InetAddress;
52
53  /**
54   * AgentGroupImpl is an RMI distributed object. This is a base class
55   * that implements the AgentGroup interfaces.
56   *
57   * Agents may belong to one or more agent coordination groups
58   * (AgentGroups), which act as focal points for distributed
59   * synchronous agent collaboration and asynchronous notifications
60   * (e.g., exceptions) among members of the agent group.
61   *
62   * @see     Agent
63   * @see     AgentGroup
64   * @author Noemi Paciorek
65   */
66
67  abstract public class AgentGroupImpl
68     extends UnicastRemoteServer
69     implements AgentGroup {
70
71     // Constants
72     /**
73      * Default timeout for collaboration in ms.
74      */
75     public static final long DEFAULT_TIMEOUT = 100000;
76     // public static final long DEFAULT_TIMEOUT = 5000;
77
78
79     // Class variables
80     protected static int groupID = 0;
81
82     // Instance variables
83     protected long timeout;          // collaboration timeout in ms
84     protected String groupName;      // name to register with RMI Name Server
85     protected Vector agentList;      // Agent IDs of group members
86     protected Vector resultList;     // results shared via collaborate
87     protected boolean collaborationInProgress;
88
89
90     //Constructors
91     /**
92      * Constructs an AgentGroupImpl object.
93      *
94      */
```

WO 98/21662 PCT/US97/20232

```
95      public AgentGroupImpl()
96          throws RemoteException {
97          this(DEFAULT_TIMEOUT);
98      }
99
100     /**
101      * Constructs an AgentGroupImpl object.
102      * @param   timeout The collaboration timeout in milliseconds
103      *
104      */
105
106     public AgentGroupImpl(long timeout)
107         throws RemoteException {
108
109         this.timeout = (timeout > 0) ? timeout : DEFAULT_TIMEOUT;
110         agentList = new Vector();
111         resultList = null;
112         collaborationInProgress = false;
113
114         /*
115          * Create a name to represent this instance of an agent group
116          * and register it with the RMI name service.
117          */
118         groupName = "AgentGroup" + nextGroupID();
119         System.out.println("groupName = " + groupName);
120
121         try {
122             Naming.rebind(groupName, this);
123         } catch (Exception e) {
124             System.out.println("AgentGroupImpl can't rebind: " + e.getMessage());
125             e.printStackTrace();
126         }
127     }
128
129
130     // Finalizer
131     protected void finalize()
132         throws Throwable {
133
134         System.out.println("AgentGroupImpl Finalizer called");
135
136         // Remove this AgentGroup's Name from the RMI registry.
137         try {
138             Naming.unbind(groupName);
139         } catch (Exception e) {
140             System.out.println("AgentGroupImpl can't unbind: " + e.getMessage());
141             e.printStackTrace();
```

```
142         }
143
144         super.finalize();
145     }
146
147
148     // Class methods
149     private final synchronized int nextGroupID() {
150         return ++groupID;
151     }
152
153
154     // Instance methods
155     /**
156      * Returns the Agent Group's unique ID.
157      * @return the Agent Group's unique ID.
158      */
159     public final String getGroupName() {
160         return groupName;
161     }
162
163
164     /**
165      * Converts an Agent Group into its string representation.
166      * @return  A String representing the Agent Group
167      */
168     public String toString() {
169         return groupName;
170     }
171
172
173     // Methods to manipulate agentList.
174
175     /**
176      * Adds an Agent to the group.
177      * @param   status  The AgentStatus object that describes the Agent.
178      */
179     public synchronized void addAgent(AgentStatus status) {
180         agentList.addElement(status);
181         System.out.println("Added Agent, ID = " + status.getAgentID() +
182             ", Status = " + status.getStatus() + " to group");
183     }
184
185     /**
186      * Locates and removes the AgentStatus object containing the specified ID.
187      * @param   agentID An Agent's unique ID.
188      * @exception NoSuchElementException    If the AgentStatus object does
```

```
189         * not exist.
190         */
191        public synchronized void removeAgent(String agentID)
192            throws NoSuchElementException, ArrayIndexOutOfBoundsException {
193
194            int index = indexOfAgent(agentID);
195            if (index == -1)
196                throw new NoSuchElementException("Can't find AgentID");
197            agentList.removeElementAt(index);
198            System.out.println("Removed Agent, ID = " + agentID + " from group");
199        }
200
201        /**
202         * Replaces the AgentStatus object for the specified ID.
203         * This method may be used to change an Agent's status, ID, or both.
204         * @param   agentID An Agent's unique ID
205         * @param   newStatus   An AgentStatus object to replace the existing one.
206         * @exception NoSuchElementException    If the AgentStatus object does
207         * not exist.
208         */
209        public synchronized void updateAgent(String agentID, AgentStatus newStatus)
210            throws NoSuchElementException, ArrayIndexOutOfBoundsException {
211
212            removeAgent(agentID);
213            addAgent(newStatus);
214            System.out.println("Updated Agent ID from " + agentID + " to New ID = " +
215                newStatus.getAgentID() + ", Status = " + newStatus.getStatus());
216        }
217
218        /*
219         * Walks the agentList looking for an AgentStatus object with the
220         * specified ID.
221         */
222        private int indexOfAgent(String agentID) {
223            int index, numAgents;
224            for (index = 0, numAgents = agentList.size(); index < numAgents; index++) {
225                AgentStatus status = (AgentStatus)agentList.elementAt(index);
226                if (agentID.compareTo(status.getAgentID()) == 0)
227                    return index;
228            }
229            return -1;
230        }
231
232        /**
233         * Determines how many agents in the group have the specified status.
234         * @param   status  The status value to search for.
235         * @return  The number of Agent's with the specified status.
```

```
236         * @see AgentStatus
237         */
238        public int agentsWithStatus(int status) {
239            int agents = 0;
240            for (Enumeration e = agentList.elements(); e.hasMoreElements(); ) {
241                AgentStatus aStatus = (AgentStatus)e.nextElement();
242                if (aStatus.getStatus() == status)
243                    agents++;
244            }
245            return agents;
246        }
247
248
249        /**
250         * Returns the number of Agents in the group.
251         * @return The number of Agents in the group.
252         */
253        public final int getGroupSize() {
254            return agentList.size();
255        }
256
257
258        // Methods to implement collaboration
259
260        /**
261          * Collaborate() is the public interface to collaboration.  It calls
262          * beginCollaboration() to synchronize the recording of agent results
263          * and to wait for all agents to arrive.
264          *
265          * If all agents in the group have arrived at the collaboration
266          * point, each agent calls the application-specific analyzeResults()
267          * method to determine the agent's next course of action.
268          *
269          * Otherwise, the agent that initiated the collaborate calls
270          * recoverCollaboration().
271          *
272          *
273          * @param      result  The result of the agent's computation.
274          * @exception  RemoteException If an error occurrs setting up network
275          * connections.
276          * @exception  AgentGroupException If the collaboration times out.
277          * @return     An Object that describes the result of collaboration
278          */
279        public Object collaborate(AgentResult result)
280            throws RemoteException, AgentGroupException {
281
282            System.out.println("collaborate RMI");
```

WO 98/21662 PCT/US97/20232

```
283
284         boolean firstCollaborator = beginCollaboration(result);
285
286         /*
287          * If collaboration failed and this is the agent that initiated
288          * collaboration, it woke up because its timer expired.  It retries
289          * once before waking up the other agents.
290          *
291          */
292         if (firstCollaborator && !collaborationSucceeded()) {
293             System.out.println("Collaboration Failed.  Retrying...");
294             synchronized(this) {
295                 try {
296                     this.wait(timeout);
297                 } catch (InterruptedException e) { }
298             }
299         }
300
301         /*
302          * If all agents in the group have arrived at the collaboration
303          * point, each agent calls the application-specific analyzeResults
304          * method to determine the agent's next course of action.
305          */
306         if (collaborationSucceeded()) {
307             System.out.println("Ready to do collaboration");
308             Object collaboration = analyzeResults(resultList.elements());
309
310             /*
311              * The agent that initiated collaboration cleans up.
312              */
313             if (firstCollaborator)
314                 endCollaboration();
315
316             return collaboration;
317         } else {
318             /*
319              * If this is the agent that originated the collaboration,
320              * it woke up because the timer expired.  It must wake up
321              * the other blocked agents (which are still sitting in
322              * beginCollaborate()) and then recover from the error.
323              */
324             if (firstCollaborator) {
325                 System.out.println("Collaboration Failed while " +
326                     agentsWithStatus(AgentStatus.INTRANSIT) +
327                     " agents were in transit");
328
329                 synchronized(this) {
```

WO 98/21662 PCT/US97/20232

```
330                     this.notifyAll();
331                 }
332                 recoverCollaboration();
333             } else
334                 System.out.println("Collaboration Failed");
335
336             throw new AgentGroupException("Collaboration timed out");
337         }
338
339     }
340
341
342     /**
343       * beginCollaboration() is called internally by collaborate() to
344       * synchronize each agent's recording of results and to wait for
345       * all agents to arrive at the collaboration point. This method acts
346       * as a distributed rendezvous, of sorts.  Each caller blocks until
347       * all agents in the group have "arrived" at the collaboration point.
348       *
349       * This method returns a boolean that indicates if the agent initiated
350       * the collaboration.
351       *
352       * @param  result      The result of the agent's computation.
353       * @exception  AgentGroupException If the group only has one member.
354       * @return A boolean indicating if the collaboration was successful.
355       *
356       */
357     protected synchronized boolean beginCollaboration(AgentResult result)
358         throws AgentGroupException {
359
360         boolean firstCollaborator = false;
361
362         System.out.println("Beginning collaboration");
363
364         if (!collaborationInProgress) {
365             /*
366              * If this agent is initiating the collaboration, allocate the
367              * resultList Vector and add result to it.  Then wait for all
368              * agents to arrive at the collaboration point or for the
369              * collaboration timer to expire.
370              */
371             if (agentList.size() < 2) {
372                 endCollaboration();
373                 throw new AgentGroupException("AgentGroup too small");
374             }
375
376             collaborationInProgress = true;
```

WO 98/21662 PCT/US97/20232

```
377                 firstCollaborator = true;
378                 resultList = new Vector();
379                 resultList.addElement(result);
380
381                 try {
382                     this.wait(timeout);
383                 } catch (InterruptedException e) { }
384
385             } else {
386
387                 /*
388                  * Add this agent's results to resultList.  If this is the
389                  * last agent to arrive, wake up the other agents.  Otherwise
390                  * wait for the rest of the agents in the group to arrive.
391                  */
392                 resultList.addElement(result);
393                 if (collaborationSucceeded())
394                     this.notifyAll();
395                 else {
396                     try {
397                         this.wait();
398                     } catch (InterruptedException e) { }
399                 }
400             }
401
402             return firstCollaborator;
403         }
404
405
406         /**
407          * Cleans up after a successful collaboration.
408          */
409         protected void endCollaboration() {
410             collaborationInProgress = false;
411         }
412
413
414         /**
415          * Cleans up after a failed collaboration.
416          */
417         protected void recoverCollaboration() {
418             endCollaboration();
419
420         }
421
422
423         /**
```

```
424         * analyzeResults() is an abstract method that must be overridden by
425         * each application.  It provides the application-specific behavior
426         * required to implement collaboration.  This method is invoked on
427         * behalf of each agent in the group.  It is passed the results
428         * computed by each agent and determines the course of action for the
429         * calling agent, if any, based on the computed results.
430         *
431         * @param  results An Enumeration of the results computed by the agents.
432         * @return Object  An application-specific object or null.
433         *
434         */
435        abstract protected Object analyzeResults(Enumeration results);
436
437        /*
438         * Private method to determine if all agents have reached the
439         * collaboration point.  This method is synchronized to prevent
440         * races with agents joining and leaving the group.
441         */
442        private synchronized boolean collaborationSucceeded() {
443            return (resultList.size() == agentList.size());
444        }
445
446
447
448    /*
449     * TURN ON LATER
450     *
451
452        // Methods to handle exceptions and other notifications
453        public void agentException(String agentID, Exception type)
454            throws RemoteException {
455        }
456
457        public void agentTermination(String agentID)
458            throws RemoteException {
459        }
460
461    */
462
463    }
```

```
1  /*  $Header: /com/meitca/hsl/zonesjagents/collaborate/AgentResult.java 4     10/01/96 3:43p Noemi $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
4   *  Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  This object holds the result of an agent's computations.
11  *
12  *      $Log: /com/meitca/hsl/zonesjagents/collaborate/AgentResult.java $
13  *
14  * 4     10/01/96 3:43p Noemi
15  * Added some javadoc comments.
16  *
17  * 3     9/06/96 3:48p Noemi
18  * Added a protected default constructor to ensure that the public
19  * constructors set up instance variables correctly.
20  *
21  * 2     8/30/96 4:22p Noemi
22  * Changed private protected instance variables to protected.
23  *
24  * 1     8/28/96 2:17p Noemi
25  * A generic class that represents the result of an agent computation.
26  * (Used by collaboration code.)
27  */
28
29  package com.meitca.hsl.zonesjagents.collaborate;
30
31  /**
32   * AgentResult contains an agent ID and a result to be shared
33   * via collaboration.
34   *
35   * @see    CollaboratorAgent
36   * @see    AgentGroupImpl
37   * @author Noemi Paciorek
38   */
39
40  public class AgentResult {
41
42    // Instance variables
43    protected String agentID;    // ID of agent that computed result
44    protected Object result;     // actual result computed
45
46    //Constructors
47    /**
```

WO 98/21662 PCT/US97/20232

```
48       * Protected Default Constructor for an AgentResult object.
49       * @exception   IllegalAccessException  Whenever called.
50       */
51      protected AgentResult()
52          throws IllegalAccessException {
53
54          System.out.println("Default constructor called for AgentResult");
55          throw new IllegalAccessException("AgentResult default constructor");
56      }
57
58      /**
59       * Constructs an AgentResult object.
60       * @param   agentID The Agent's unique ID.
61       * @param   result  The result to be shared via collaboration.
62       */
63      public AgentResult(String agentID, Object result) {
64          this.agentID = agentID;
65          this.result = result;
66      }
67
68
69      // Instance methods
70      /**
71       * Gets an Agent's unique ID from this AgentResult object.
72       * @return  An Agent's unique ID.
73       */
74      public final String getAgentID() {
75          return agentID;
76      }
77
78      /**
79       * Gets the result to be shared via collaboration
80       * from this AgentResult object.
81       * @return The result to be shared via collaboration.
82       */
83      public Object getResult() {
84          return result;
85      }
86
87  }
```

WO 98/21662 PCT/US97/20232

```
1  /*  $Header: /com/meitca/hsl/zonesjagents/collaborate/AgentStatus.java 2     10/01/96 3:44p Noemi $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
4   *  Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  This object holds an Agent's status (e.g. ACTIVE).
11  *
12  *      $Log: /com/meitca/hsl/zonesjagents/collaborate/AgentStatus.java $
13  *
14  * 2     10/01/96 3:44p Noemi
15  * Added some javadoc comments.
16  *
17  * 1     9/06/96 3:31p Noemi
18  * This class represents an agent's current status (e.g., ACTIVE).
19  */
20
21  package com.meitca.hsl.zonesjagents.collaborate;
22
23  /**
24   * AgentStatus contains an agent ID and a status value.  Each AgentGroup
25   * contains an AgentStatus object for each Agent in the group.
26   *
27   * @see    CollaboratorAgent
28   * @see    AgentGroupImpl
29   * @author Noemi Paciorek
30   */
31
32  public class AgentStatus {
33
34    // Constants
35    private static final int FIRST_STATUS = 1;
36    /**
37     * Agent is active.
38     */
39    public static final int ACTIVE = 1;
40    /**
41     * Agent is intransit.
42     */
43    public static final int INTRANSIT = 2;
44    /**
45     * Agent is in an unknown state.
46     */
47    public static final int UNKNOWN = 3;
```

WO 98/21662 PCT/US97/20232

```
48      private static final int LAST_STATUS = UNKNOWN;
49
50      // Instance variables
51      protected String agentID;    // ID of agent
52      int              status;     // current status
53
54      //Constructors
55      /**
56       * Constructs an AgentStatus object.
57       * @exception   IllegalAccessException  Whenever called.
58       */
59      protected AgentStatus()
60          throws IllegalAccessException {
61
62          System.out.println("Default constructor called for AgentStatus");
63          throw new IllegalAccessException("AgentStatus default constructor");
64      }
65
66
67      /**
68       * Constructs an AgentStatus object.
69       * @param   agentID The Agent's unique ID.
70       */
71      public AgentStatus(String agentID) {
72          this(agentID, ACTIVE);
73      }
74
75
76      /**
77       * Constructs an AgentStatus object.
78       * @param   agentID The Agent's unique ID.
79       * @param   status  The Agent's status.
80       */
81      public AgentStatus(String agentID, int status) {
82          this.agentID = agentID;
83          this.status = ((status < FIRST_STATUS) || (status > LAST_STATUS)) ?
84              UNKNOWN : status;
85      }
86
87
88      // Instance methods
89      /**
90       * Extracts an Agent's unique ID from an AgentStatus object.
91       * @return An Agent's unique ID.
92       */
93      public final String getAgentID() {
94          return agentID;
```

WO 98/21662 PCT/US97/20232

```
95      }
96
97      /**
98       * Extracts an Agent's current status from an AgentStatus object.
99       * @return An Agent's current status.
100      */
101     public final int getStatus() {
102         return status;
103     }
104
105 }
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/collaborate/CollaboratorAgent.java 5     10/01/96 3:45p Noemi $
 *
 *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
 *  Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Derived class for mobile Agent collaboration
 *
 *      $Log: /com/meitca/hsl/zonesjagents/collaborate/CollaboratorAgent.java $
 *
 * 5     10/01/96 3:45p Noemi
 * Added some javadoc comments.
 *
 * 4     9/20/96 4:16p Noemi
 * Changed finalizer to ensure that it performs cleanup no more than once.
 * ( A bug in the garbage collector may cause the finalizer to be called
 * more than once!)
 *
 * 3     9/10/96 5:08p Noemi
 * Renamed setIntransitStatus and setActiveStatus to prepareForTransport
 * and completedTransport, respectively.
 *
 * 2     9/06/96 3:57p Noemi
 * Added the following methods: addGroup, removeGroup, updateStatus,
 * setIntransitStatus, and setActiveStatus.
 *
 * 1     9/04/96 3:15p Noemi
 * Subclass of Agent that implements a collaborating Agent.
 *
 */
package com.meitca.hsl.zonesjagents.collaborate;

import java.util.*;
import java.net.InetAddress;

import com.meitca.hsl.zonesjagents.shared.Agent;

/**
 * A subclass of Agent that enables collaboration among agents in a group.
 * Agents wishing to collaborate must extend this class to do some useful work.
 * They must also belong to an agent coordination group (AgentGroup).  Generally,
 * an application first creates an AgentGroup.  Then it obtains a reference to
 * the RMI interface for the group by looking up the group in the local RMI name
 * service. Subsequently, it creates new CollaboratorAgents and populates the
```

```
48   * group with them (by passing the contructor a reference to the interface).
49   *
50   * @see     Agent
51   * @see     AgentGroup
52   * @author  Noemi Paciorek
53   */
54  public abstract class CollaboratorAgent
55      extends Agent {
56
57          // Instance variables
58          protected Vector groupList;             // AgentGroups this agent belongs to
59
60          // Constructors
61          /**
62           * Constructs a CollaboratorAgent
63           * @exception   IllegalAccessException  Whenever called.
64           */
65      protected CollaboratorAgent()
66          throws IllegalAccessException {
67
68          System.out.println("Default constructor called for CollaboratorAgent");
69          throw new IllegalAccessException("CollaboratorAgent Default constructor");
70      }
71
72          /**
73           * Constructs a CollaboratorAgent
74           * @param   group   The AgentGroup to which this Agent belongs.
75           */
76          public CollaboratorAgent(AgentGroup group) {
77              /*
78               * Store a copy of the agentID in the AgentGroup and save a
79               * reference to the AgentGroup in groupList.
80               */
81              groupList = new Vector();
82              addGroup(group);
83          }
84
85
86          /**
87           * Constructs a CollaboratorAgent
88           * @param   groups   A list of AgentGroups to which this Agent belongs.
89           */
90          public CollaboratorAgent(Vector groups) {
91              this(groups, null);
92          }
93
94
```

WO 98/21662 PCT/US97/20232

```
95          /**
96           * Constructs a CollaboratorAgent
97           * @param   groups  A list of AgentGroups to which this Agent belongs.
98           * @param   oldID   The unique ID of an agent to be replaced by this agent.
99           */
100         public CollaboratorAgent(Vector groups, String oldID) {
101             /*
102              * Save a copy of agentID in each of the associated AgentGroups and
103              * store a reference to each AgentGroup in groupList.
104              */
105         groupList = new Vector();
106             for (Enumeration e = groups.elements(); e.hasMoreElements(); ) {
107                 AgentGroup group = (AgentGroup)e.nextElement();
108                 groupList.addElement(group);
109
110                 try {
111                     if (oldID == null)
112                         group.addAgent(new AgentStatus(agentID));
113                     else
114                         group.updateAgent(oldID, new AgentStatus(agentID));
115                     System.out.println("Group size = " + group.getGroupSize());
116                 } catch (Exception ex) {
117                     System.out.println("Agent constructor: can't add agent to group: " +
118                         ex.getMessage());
119                     System.out.println("AgentID = " + agentID + ", oldID = " + oldID);
120                     ex.printStackTrace();
121                 }
122
123             }
124         }
125
126
127         // Finalizer
128         protected void finalize()
129         throws Throwable {
130
131         System.out.println("CollaboratorAgent Finalizer called");
132
133             /*
134              * Remove agentID from all groups this agent belongs to.
135              */
136             if (groupList != null) {
137                 for (Enumeration e = groupList.elements(); e.hasMoreElements(); ) {
138                     AgentGroup group = (AgentGroup)e.nextElement();
139                     try {
140                         group.removeAgent(agentID);
141                     } catch (Exception ex) {
```

WO 98/21662 PCT/US97/20232

```
142                         System.out.println("CollaboratorAgent finalizer: can't remove agent from group: " +
143                             ex.getMessage());
144                         ex.printStackTrace();
145                     }
146                 }
147                 groupList = null; // in case finalizer gets called twice (it has been!)
148             }
149                 super.finalize();
150             }
151
152             // Instance methods
153             /**
154              * Returns a list of groups the CollaboratorAgent belongs to.
155              * @return The list of groups this CollaboratorAgent belongs to.
156              */
157             public final synchronized Enumeration getGroups() {
158                 return groupList.elements();
159             }
160
161             /**
162              * Add a group to this agent's groupList and add the agent
163              * to the group's agent list.
164              * @param   group   The group to be added.
165              */
166             public synchronized void addGroup(AgentGroup group) {
167                 groupList.addElement(group);
168
169                 try {
170                     group.addAgent(new AgentStatus(agentID));
171                     System.out.println("Group size = " + group.getGroupSize());
172                 } catch (Exception ex) {
173                     System.out.println("Can't add agent to group: " + ex.getMessage());
174                     ex.printStackTrace();
175                 }
176             }
177
178             /**
179              * Remove a group from this agent's groupList and remove the agent
180              * from the group's agent list.
181              * @param   group   The group to be removed.
182              */
183             public synchronized void removeGroup(AgentGroup group) {
184                 groupList.removeElement(group);
185
186                 try {
187                     group.removeAgent(agentID);
188                 } catch (Exception ex) {
```

```
189                     System.out.println("Can't remove agent from group: " + ex.getMessage());
190                     ex.printStackTrace();
191                 }
192             }
193
194
195             /**
196              * Update an agent's status in all groups it belongs to.
197              * @param   status  The new status.
198              */
199             public synchronized void updateStatus(int status) {
200                 if (groupList != null)
201                     for (Enumeration e = groupList.elements(); e.hasMoreElements(); ) {
202                         AgentGroup group = (AgentGroup)e.nextElement();
203                         try {
204                             group.updateAgent(agentID, new AgentStatus(agentID, status));
205                         } catch (Exception ex) {
206                             System.out.println("CollaboratorAgent finalizer: can't remove agent from group: " +
207                                 ex.getMessage());
208                             ex.printStackTrace();
209                         }
210                 }
211         }
212
213         /**
214          * Set an agent's status to INTRANSIT before transport.
215          */
216         public void prepareForTransport() {
217             updateStatus(AgentStatus.INTRANSIT);
218         }
219
220
221         /**
222          * Set an agent's status to ACTIVE after transport.
223          */
224         public void completedTransport() {
225             updateStatus(AgentStatus.ACTIVE);
226         }
227
228 }
```

```
1  /*  $Header: /com/meitca/hsl/zonesjagents/event/EventException.java 1      10/14/96 6:55p Noemi $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
4   *  Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  Event exceptions
11  *
12  *      $Log: /com/meitca/hsl/zonesjagents/event/EventException.java $
13  *
14  * 1     10/14/96 6:55p Noemi
15  * Base class for exceptions generated during event processing.
16  */
17 package com.meitca.hsl.zonesjagents.event;
18
19 /**
20  * Exceptions encountered by Events.
21  *
22  * @see EventType
23  * @author  Noemi Paciorek
24  */
25
26 public class EventException extends Exception {
27
28     // Constructors
29     /**
30      * Constructs an EventException.
31      * @exception   IllegalAccessException  Whenever called.
32      */
33     protected EventException()
34         throws IllegalAccessException {
35
36         System.out.println("Default constructor called for EventException");
37         throw new IllegalAccessException("EventException Default constructor");
38     }
39
40     /**
41      * Constructs an EventException.
42      * @param   type    A String describing the exception.
43      */
44     public EventException(String type) {
45         super(type);
46     }
47 }
```

```
 1  /*  $Header: /com/meitca/hsl/zonesjagents/event/EventGroup.java 1     10/14/96 6:59p Noemi $
 2   *
 3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
 4   *  Information Technology Center America.
 5   *      All rights reserved.
 6   *
 7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 8   *
 9   *  DESCRIPTION
10   *  Event Group distributed interface.
11   *
12   *      $Log: /com/meitca/hsl/zonesjagents/event/EventGroup.java $
13   *
14   * 1     10/14/96 6:59p Noemi
15   * Event Group interface.
16   *
17   */
18
19  package com.meitca.hsl.zonesjagents.event;
20
21  import java.util.NoSuchElementException;
22
23  import java.rmi.Remote;
24  import java.rmi.RemoteException;
25
26  /**
27   * EventGroup is a distributed RMI interface for event groups.
28   *
29   * An EventGroup consists of a group of objects that wish to receive
30   * events generated by the other members of the group. The EventGroup
31   * serves as a gateway between the objects in the group and forwards
32   * events posted to it to the members of the group.
33   *
34   * The members of the group must all implement the EventPost interface.
35   *
36   * @see EventGroupImpl
37   * @author Noemi Paciorek
38   */
39
40  public interface EventGroup
41      extends Remote {
42
43      /**
44       * Adds a member to the event group.
45       * @param   object  An Object that implements the EventPost interface.
46       * @exception  RemoteException  If an error occurs setting up network
47       * connections.
```

WO 98/21662 PCT/US97/20232

```
48          * @exception  ClassCastException  If object does not implement EventPost.
49          */
50         public void addMember(Object object)
51             throws RemoteException, ClassCastException;
52
53         /**
54          * Removes a member from the group
55          * @param   object  The Object to remove
56          * @exception  RemoteException  If an error occurrs setting up network
57          * connections
58          * @exception  NoSuchElementException  If object is not a member of the group.
59          */
60         public void removeMember(Object object)
61             throws RemoteException, NoSuchElementException;
62
63     }
```

```
1  /*  $Header: /com/meitca/hsl/zonesjagents/event/EventGroupImpl.java 2     10/22/96 7:10p Noemi $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
4   *  Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  Event Group distributed object.
11  *
12  *      $Log: /com/meitca/hsl/zonesjagents/event/EventGroupImpl.java $
13  *
14  * 2     10/22/96 7:10p Noemi
15  * Changed the throws clause of the postEvent method.
16  *
17  * 1     10/14/96 8:59p Noemi
18  * Event group implementation.
19  *
20  */
21
22  package com.meitca.hsl.zonesjagents.event;
23
24
25  import java.util.*;
26  import java.io.*;
27
28  import java.rmi.server.UnicastRemoteServer;
29  import java.rmi.RemoteException;
30  import java.rmi.Naming;
31  import java.rmi.AlreadyBoundException;
32
33  import com.meitca.hsl.util.UniqueID;
34
35  /**
36   * EventGroupImpl is an RMI distributed object that implements
37   * Event Group and EventPost.
38   *
39   * An EventGroup consists of a group of objects that wish to receive
40   * events generated by the other members of the group. The EventGroup
41   * serves as a gateway between the objects in the group and forwards
42   * events posted to it to the members of the group.
43   *
44   * The members of the group must all implement the EventPost interface.
45   *
46   * @see EventGroup
47   * @author Noemi Paciorek
```

```
48   */
49
50   public class EventGroupImpl
51       extends UnicastRemoteServer
52       implements EventGroup, EventPost {
53
54       // Instance variables
55
56       /** A list of the group's members. */
57       protected Vector memberList;
58
59       // Constructors
60       /**
61        * Constructs an EventGroupImpl object.
62        * @exception  RemoteException  If an error occurrs setting up network
63        * connections.
64        */
65       public EventGroupImpl()
66           throws RemoteException {
67           /*
68            * Create member list and event queue.
69            */
70           memberList = new Vector();
71       }
72
73       // Instance methods
74
75       // EventGroup methods
76      /**
77        * Add a member to the event group.
78        * @param   object  An Object that implements the EventPost interface.
79        * @exception  ClassCastException  If object does not implement EventPost.
80        */
81       public synchronized void addMember(Object object)
82           throws ClassCastException {
83
84           try {
85               EventPost eventPost = (EventPost)object;
86           } catch (ClassCastException e) {
87               System.out.println("addMember: Object does not implement Event Post");
88               throw e;
89           }
90           memberList.addElement(object);
91       }
92
93
94       /**
```

```
95        * Remove a member from the group
96        * @param   object  The Object to remove
97        * @exception  NoSuchElementException  If object is not a member of the group.
98        */
99       public synchronized void removeMember(Object object)
100          throws NoSuchElementException {
101
102          if (!memberList.removeElement(object))
103              throw new NoSuchElementException("Object not member of group");
104      }
105
106
107      // EventPost methods
108      /**
109       * Forward an event to all members of the group.
110       * @param   event   The event to post.
111       * @exception   EventException  If an error occurs posting the event.
112       */
113      public void postEvent(EventType event)
114          throws RemoteException, EventException, IOException {
115
116              for (Enumeration e = memberList.elements(); e.hasMoreElements(); ) {
117                  EventPost eventQueue = (EventPost)e.nextElement();
118                  eventQueue.postEvent(event);
119              }
120      }
121
122  }
```

```
1  /*  $Header: /com/meitca/hsl/zonesjagents/event/EventHandler.java 1     10/14/96 6:56p Noemi $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
4   *  Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  Event handler interface
11  *
12  *  $Log: /com/meitca/hsl/zonesjagents/event/EventHandler.java $
13  *
14  * 1     10/14/96 6:56p Noemi
15  * Event handler interface.
16  */
17
18 package com.meitca.hsl.zonesjagents.event;
19
20 import java.rmi.Remote;
21 import java.rmi.RemoteException;
22
23 /**
24  * EventHandler is a distributed RMI interface for handling events.
25  *
26  * @author Noemi Paciorek
27  */
28
29 public interface EventHandler {
30
31     // Instance methods.
32     /**
33      * Event handler interface.
34      * @param   event   Current event.
35      * @exception   RemoteException If an error occurrs setting up network
36      * connections.
37      * @exception   EventException  If an error occurs handling the event.
38      */
39      public void handleEvent(EventType event)
40         throws RemoteException, EventException;
41 }
42
```

```
 1  /*  $Header: /com/meitca/hsl/zonesjagents/event/EventManager.java 2     10/22/96 7:12p Noemi $
 2   *
 3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
 4   *  Information Technology Center America.
 5   *      All rights reserved.
 6   *
 7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 8   *
 9   *  DESCRIPTION
10   *  Event Manager Interface
11   *
12   *  $Log: /com/meitca/hsl/zonesjagents/event/EventManager.java $
13   *
14   * 2     10/22/96 7:12p Noemi
15   * Changed registerEvents and unregisterEvents to take an array of
16   * Strings, rather than an array of EventType.  Also changed several of
17   * the throws clauses.
18   *
19   * 1     10/14/96 6:58p Noemi
20   * Event Manager interface.
21   */
22
23  package com.meitca.hsl.zonesjagents.event;
24
25  import java.io.*;
26  import java.rmi.Remote;
27  import java.rmi.RemoteException;
28
29  /**
30   * EventManager is a distributed RMI interface for event management.
31   *
32   * @see EventManagerImpl
33   * @author Noemi Pachorek
34   */
35
36  public interface EventManager
37      extends Remote {
38
39      // Instance methods.
40      // Methods used to register interest in specified events.
41
42      /**
43       * Register interest in receiving specified events.
44       * @param   events      An array of the names of events that an object
45       * is interested in receiving.
46       * @param   eventQueue  The associated event queue's EventPost interface.
47       * @exception  RemoteException If an error occurrs setting up network
```

WO 98/21662 PCT/US97/20232

```
48       * connections.
49       */
50      public void registerEvents(String events[], EventPost eventQueue)
51          throws RemoteException, IOException;
52
53      /**
54       * Register interest in receiving all events posted to this EventManager.
55       * @param   eventQueue  The associated event queue's EventPost interface.
56       * @exception  RemoteException If an error occurrs setting up network
57       * connections.
58       */
59      public void registerAll(EventPost eventQueue)
60          throws RemoteException, IOException;
61
62      // Methods to unregister events.
63      /**
64       * Remove registration for specified events.
65       * @param   events      An array of the names of events that an object
66       * is no longer interested in receiving.
67       * @param   eventQueue  The associated event queue's EventPost interface.
68       * @exception   RemoteException If an error occurrs setting up network
69       * connections.
70       * @exception   EventManagerException If the event queue was not registered
71       * for one or more of the specified exceptions.
72       */
73      public void unregisterEvents(String events[], EventPost eventQueue)
74          throws RemoteException, IOException, EventManagerException;
75
76      /**
77       * Remove registration for all events.
78       * @param   eventQueue  The associated event queue's EventPost interface.
79       * @exception   RemoteException If an error occurrs setting up network
80       * connections.
81       */
82      public void unregisterAll(EventPost eventQueue)
83          throws RemoteException, IOException;
84
85  }
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/event/EventManagerException.java 1     10/14/96 6:57p Noemi $
 *
 *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
 *  Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  EventManager exceptions.
 *
 *      $Log: /com/meitca/hsl/zonesjagents/event/EventManagerException.java $
 *
 * 1     10/14/96 6:57p Noemi
 * Exceptions generated by the Event Manager.
 */
package com.meitca.hsl.zonesjagents.event;

/**
 * Exceptions encountered by the EventManager.
 *
 * @see EventMangerImpl
 * @author  Noemi Paciorek
 */

public class EventManagerException extends EventException {

    // Constructors
    /**
     * Constructs an EventManagerException.
     * @exception   IllegalAccessException  Whenever called.
     */
    protected EventManagerException()
        throws IllegalAccessException {

        System.out.println("Default constructor called for EventManagerException");
        throw new IllegalAccessException("EventManagerException Default constructor");
    }

    /**
     * Constructs an EventManagerException.
     * @param   type    A String describing the exception.
     */
    public EventManagerException(String type) {
        super(type);
    }
}
```

EventManagerException.java_1 Mon Nov 11 16:26:41 1996 2

```
48
49
50
51
52
```

```
/*  $Header: /com/meitca/hsl/zonesjagents/event/EventManagerImpl.java 3     10/24/96 4:57p Noemi $
 *
 *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
 *  Information Technology Center America.
 *      All rights reserved.
 *
 *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 *
 *  DESCRIPTION
 *  Event Manager distributed object.
 *
 *  $Log: /com/meitca/hsl/zonesjagents/event/EventManagerImpl.java $
 *
 * 3     10/24/96 4:57p Noemi
 * Added support for an Event Manager properties file and properties that
 * specify if persistence is enabled and the name of the persistent
 * storage file.
 *
 * 2     10/22/96 7:28p Noemi
 * Save full and specific event registrations to persistent storage.
 * Constructor reads persistent data and reconstructs notifications.
 * EventManager removes event queues that it can't post events to.
 * Changed registerEvents and unregisterEvents to take an array of String.
 * Made constructor protected and added a main method.
 * Perform cleanup when receiving conflicting registrations (full vs.
 * specific).
 *
 * 1     10/14/96 6:58p Noemi
 * The Event Manager.
 */

package com.meitca.hsl.zonesjagents.event;

import java.util.*;
import java.io.*;

import java.rmi.server.UnicastRemoteServer;
import java.rmi.RemoteException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.AlreadyBoundException;

import com.meitca.hsl.zonesjagents.shared.*;

import com.meitca.hsl.zonesjagents.security.InsecurityManager;
```

WO 98/21692 PCT/US97/20232

```
48  /**
49   * EventManagerImpl is an Event Manager that implements the RMI
50   * EventManager and EventPost interfaces.
51   *
52   * This class allows objects to register their interest in receiving
53   * specified events and to delete their registrations.
54   *
55   * Objects post events by calling the EventManager's postEvent method
56   * (defined by the EventPost interface).
57   *
58   * The EventManager is a sentinel object (i.e., there is only one
59   * per system.
60   *
61   * @see EventManager
62   * @see EventPost
63   * @see EventHandler
64   * @see EventManagerProxy
65   * @author  Noemi Paciorek
66   */
67
68  public class EventManagerImpl
69      extends UnicastRemoteServer
70      implements EventManager, EventPost {
71
72      // Constants
73      private static final String EVENT_MANAGER_NAME = "EventManager";
74
75      private static final String PROPERTIES_DESCRIPTION = "Properties File for EventManager";
76          private static final String ENABLE_PERSISTENCE = "EventManager.persistence.Enable";
77          private static final String PERSISTENCE_FILE = "EventManager.persistence.FileName";
78      // private static final String DEFAULT_PERSISTENCE_FILE = "C:\\temp\\EventManager.store";
79      private static final String DEFAULT_PERSISTENCE_FILE = "EventManager.store";
80
81
82      /* IDs for persistent objects */
83      private static final int EVENT_REGISTRATION_ID = 1;
84      private static final int FULL_REGISTRATION_ID = 2;
85
86      private static final int STORAGE_PADDING = 1024;
87
88
89      // Instance variables
90
91      /**
92       * Hash table that maps class names derived from EventType to
93       * registered EventPost stubs.
94       */
```

```
95      protected Hashtable eventRegistrations;
96
97      /** List of EventPost stubs registered to receive all notifications. */
98      protected Vector fullRegistrations;
99
100     /**
101      * Hash table that maps an EventPost stub to the list of events it
102      * is registered for. (Actually, it maps the stub to the class name of
103      * the event).
104      */
105     protected Hashtable eventNotifications;
106
107     /** A reference to this EventManager's persistent store handler. */
108     protected PersistentStoreHandler persistenceManager;
109
110     /** Name of the persistent storage file */
111     protected String persistentStoreFile;
112
113     /** The properties file */
114     protected JASProperties properties;
115
116     // Constructors
117     /**
118      * Constructs an EventManager.  EventManagerImpl has no public
119      * constructors. Applications wishing to utilize the Event Manager's
120      * services must access it via EventManagerProxy.
121      */
122     protected EventManagerImpl()
123         throws Exception {
124
125         /*
126          * Locate properties file.
127          */
128                 try {
129                         properties = new JASProperties(EVENT_MANAGER_NAME,
130                             JASProperties.SERVER, PROPERTIES_DESCRIPTION, false);
131                         System.out.println("EventManagerImpl: Located properties file");
132
133                 } catch (Exception e) {
134                         System.out.println("EventManagerImpl: Couldn't access Properties file for Conduit Ser
135                         System.out.println("\tContinuing with defaults.");
136                         // Construct an empty Properties object.
137                         properties = new JASProperties();
138                 }
139
140         /*
141          * Recover event and full registrations from persistent storage.
```

```
142             * if required. Then reconstruct event notifications.
143             */
144            recoverRegistrations();
145
146            /*
147             * Register this instance with the RMI registry.
148             */
149            try {
150                Naming.rebind(EVENT_MANAGER_NAME, this);
151            } catch (Exception e) {
152                System.out.println("EventManagerImpl can't bind " + EVENT_MANAGER_NAME + ": " + e.getMessage());
153                e.printStackTrace();
154                throw e;
155            }
156        }
157
158
159        // Finalizer
160        /**
161         * Removes the EventManager's entry from the RMI registry.
162         */
163        protected void finalize()
164            throws Throwable {
165
166            System.out.println("EventManagerImpl Finalizer called");
167
168            // Remove this EventManager's Name from the RMI registry.
169            try {
170                Naming.unbind(EVENT_MANAGER_NAME);
171            } catch (Exception e) {
172                System.out.println("EventManagerImpl can't unbind: " + e.getMessage());
173                e.printStackTrace();
174            }
175
176            // Delete reference to Persistent Store Manager.
177            persistenceManager = null;
178
179            super.finalize();
180        }
181
182
183
184        // Instance methods
185
186        // Methods to add event registrations.
187        /**
188         * Register interest in receiving specified events.
```

WO 98/21662 PCT/US97/20232

```
189       * To remove these registrations use
190       * unregisterEvents(EventType events[], EventPost eventQueue).
191       * @param   events      An array of the names of events that an object
192       * is interested in receiving.
193       * @param   eventQueue  The associated event queue's EventPost interface.
194       */
195      public synchronized void registerEvents(String events[], EventPost eventQueue)
196          throws IOException {
197
198          /*
199           * If the event queue has already registered to receive all
200           * notifications, ignore this request.
201           */
202          if (fullRegistrations.contains(eventQueue))
203              return;
204
205          /*
206           * Get the list of events the queue has registered
207           * for notification.  If the list does not exist, create it.
208           */
209          Vector notifyList = (Vector)eventNotifications.get(eventQueue);
210          if (notifyList == null) {
211              notifyList = new Vector();
212              eventNotifications.put(eventQueue, notifyList);
213          }
214
215          for (int i = 0; i < events.length; i++) {
216              String eventName = events[i];
217              System.out.println("registerEvents, event = " + eventName);
218
219              /*
220               * Add the event to the queue's list of registered events.
221               */
222              if (!notifyList.contains(eventName)) {
223                  System.out.println("registerEvents, adding event name to notify list");
224                  notifyList.addElement(eventName);
225              }
226
227              /*
228               * Look up the event name in the eventRegistrations hash table
229               * and add the queue to the list of registered Objects for the event.
230               */
231              Vector registryList = (Vector)eventRegistrations.get(eventName);
232              if (registryList == null) {
233                  registryList = new Vector();
234                  eventRegistrations.put(eventName, registryList);
235                  System.out.println("registerEvents, adding event queue to registry list");
```

WO 98/21662 PCT/US97/20232

```
236                 registryList.addElement(eventQueue);
237
238             } else if (!registryList.contains(eventQueue)) {
239                 System.out.println("registerEvents, adding event queue to registry list");
240                 registryList.addElement(eventQueue);
241             }
242         }
243         /*
244          * Write the updated eventRegistrations table to
245          * persistent storage.
246          */
247         updateEventRegistrations();
248     }
249
250
251     /**
252      * Register interest in receiving all events posted to this EventManager.
253      * To remove these registrations use
254      * unregisterAll(EventPost eventQueue).
255      * @param   eventQueue  The associated event queue's EventPost interface.
256      */
257     public synchronized void registerAll(EventPost eventQueue)
258         throws IOException {
259
260         System.out.println("registerAll");
261
262         /*
263          * If this event queue already registered to receive
264          * specific registrations, remove them.
265          */
266         if ((Vector)eventNotifications.get(eventQueue) != null)
267             unregisterAll(eventQueue);
268
269         /*
270          * If the event queue has already been registered for
271          * all notifications, ignore this request.
272          * Otherwise add the queue the the list of full registrations.
273          */
274         if (!fullRegistrations.contains(eventQueue)) {
275             System.out.println("registerAll, adding event queue to full registrations");
276             fullRegistrations.addElement(eventQueue);
277             /*
278              * Write the updated fullRegistrations list to
279              * persistent storage.
280              */
281             updateFullRegistrations();
282         }
```

WO 98/21662 PCT/US97/20232

```
283     }
284
285
286     // Methods to remove registrations.
287
288     /**
289      * Remove registration for specified events.
290      * @param   events      An array of the names of events that an object
291      * is no longer interested in receiving.
292      * @param   eventQueue  The associated event queue's EventPost interface.
293      * @exception   EventManagerException If the event queue was not registered
294      * for one or more of the specified exceptions.
295      */
296     public synchronized void unregisterEvents(String events[], EventPost eventQueue)
297         throws EventManagerException, IOException {
298
299         /*
300          * Get the list of events the queue has registered
301          * for notification.
302          */
303         Vector notifyList = (Vector)eventNotifications.get(eventQueue);
304         if (notifyList == null)
305             throw new EventManagerException("Bad event queue");
306
307         for (int i = 0; i < events.length; i++) {
308             String eventName = events[i];
309             System.out.println("unregisterEvents, event = " + eventName);
310
311             /*
312              * Remove the event from the queue's list of registered events.
313              */
314             System.out.println("unregisterEvents, removing event name from notify list");
315             notifyList.removeElement(eventName);
316
317             /*
318              * Look up the event name in the eventRegistrations hash table
319              * and remove the queue from the list of registered Objects for
320              * the event. If the list is empty, remove the event's hash
321              * table entry.
322              */
323             Vector registryList = (Vector)eventRegistrations.get(eventName);
324             if (registryList == null)
325                 throw new EventManagerException("Bad event");
326
327             System.out.println("unregisterEvents, removing event queue from registry list");
328             registryList.removeElement(eventQueue);
329
```

```
330             if (registryList.isEmpty()) {
331                 System.out.println("unregisterEvents, removing hash table entry for event " + eventName);
332                 eventRegistrations.remove(eventName);
333             }
334         }
335
336         /*
337          * Write the updated eventRegistrations table to
338          * persistent storage.
339          */
340         updateEventRegistrations();
341
342         /*
343          * If the queue's list of registered events is empty, remove
344          * the queue's hash table entry.
345          */
346         if (notifyList.isEmpty()) {
347             System.out.println("unregisterEvents, removing event queue's hash table entry");
348             eventNotifications.remove(eventQueue);
349         }
350     }
351
352
353     /**
354      * Remove registration for all events.
355      * @param   eventQueue  The associated event queue's EventPost interface.
356      * This method is often called by finalizers, so it ignores errors.
357      */
358     public synchronized void unregisterAll(EventPost eventQueue)
359         throws IOException {
360
361         if (fullRegistrations.contains(eventQueue)) {
362             /*
363              * If the event queue was registered to receive all registrations,
364              * remove it from the fullRegistrations hash table.
365              */
366             System.out.println("unregisterAll, removing full registration");
367             fullRegistrations.removeElement(eventQueue);
368
369             /*
370              * Write the updated fullRegistrations list to
371              * persistent storage.
372              */
373             updateFullRegistrations();
374         } else {
375             /*
376              * Locate the event queue's list of registered events.
```

```
377              */
378             Vector notifyList = (Vector)eventNotifications.get(eventQueue);
379             if (notifyList == null)
380                 return;
381
382             System.out.println("unregisterAll, removing event registrations");
383
384                 for (Enumeration e = notifyList.elements(); e.hasMoreElements(); ) {
385                     String eventName = (String)e.nextElement();
386                 System.out.println("unregisterAll, event = " + eventName);
387
388                 Vector registryList = (Vector)eventRegistrations.get(eventName);
389                 if (registryList == null)
390                     continue;
391                 /*
392                  * For each event in the list, remove the queue from the
393                  * event's list of registered Objects.
394                  * If the list is empty, remove the event's hash table entry.
395                  */
396                 System.out.println("removing event queue from registry list");
397                 registryList.removeElement(eventQueue);
398                 if (registryList.isEmpty()) {
399                     System.out.println("unregisterAll, removing hash table entry for event " + eventName);
400                     eventRegistrations.remove(eventName);
401                     registryList = null;
402                 }
403                 }
404
405             /*
406              * Write the updated eventRegistrations table to
407              * persistent storage.
408              */
409             updateEventRegistrations();
410
411             /*
412              * Remove the event queue's hash table entry.
413              */
414             System.out.println("unregisterAll, removing event queue's hash table entry");
415             eventNotifications.remove(eventQueue);
416         }
417     }
418
419
420     // Methods to post events and send notifications
421
422     /**
423      * Notify objects that an event occurred (EventPost interface).
```

```
424       * @param   event   Event being posted.
425       * @exception   EventException  If an error occurs posting the event.
426       */
427      public synchronized void postEvent(EventType event)
428          throws RemoteException, EventException, IOException {
429
430          System.out.println("EventManager, posting event");
431          /*
432           * Handle specific registrations for this event. Locate this event's
433           * entry in the eventRegistrations hash table and post the event to
434           * each of the event queues on its hash chain.
435           */
436          String eventName = event.getClass().getName();
437          System.out.println("postEvent, event = " + eventName);
438
439          System.out.println("postEvent, checking specific registrations");
440
441          /*
442           * Post the event to the event queues registered to
443           * receive it.
444           */
445          Vector badEventRegistrations = new Vector();
446
447          Vector registryList = (Vector)eventRegistrations.get(eventName);
448          if (registryList != null) {
449              System.out.println("registryList contains " + registryList.size() +
450                  " elements");
451
452                  for (Enumeration e = registryList.elements(); e.hasMoreElements(); ) {
453                      EventPost eventQueue = (EventPost)e.nextElement();
454
455                      System.out.println("Posting event");
456
457                      try {
458                          eventQueue.postEvent(event);
459                      } catch (Exception e2) {
460                          System.out.println("Event post failed");
461                          badEventRegistrations.addElement(eventQueue);
462                      }
463                  }
464              }
465
466              /*
467               * Post the event to all Objects that have requested notification
468               * of all events.
469               */
470              System.out.println("postEvent, checking full registrations");
```

```
471             System.out.println("fullRegistrations contains " + fullRegistrations.size() +
472                 " elements");
473
474             Vector badFullRegistrations = new Vector();
475             for (Enumeration e = fullRegistrations.elements(); e.hasMoreElements(); ) {
476                 EventPost eventQueue = (EventPost)e.nextElement();
477
478                 System.out.println("Posting event");
479                 try {
480                     eventQueue.postEvent(event);
481                 } catch (Exception e3) {
482                     System.out.println("Event post failed");
483                     badFullRegistrations.addElement(eventQueue);
484                 }
485             }
486
487             /*
488              * If the EventManager was unable to contact any event queues,
489              * remove them from eventRegistrations and/or fullRegistrations
490              * and update the affected registrations in persistent storage.
491              */
492         if (!badEventRegistrations.isEmpty()) {
493             for (Enumeration e = badEventRegistrations.elements(); e.hasMoreElements(); ) {
494                 EventPost eventQueue = (EventPost)e.nextElement();
495                 System.out.println("Removing bad event queue from registry list");
496                 registryList.removeElement(eventQueue);
497             }
498             updateEventRegistrations();
499         }
500             badEventRegistrations = null;
501
502         if (!badFullRegistrations.isEmpty()) {
503             for (Enumeration e = badFullRegistrations.elements(); e.hasMoreElements(); ) {
504                 EventPost eventQueue = (EventPost)e.nextElement();
505                 System.out.println("Removing bad event queue from full list");
506                 fullRegistrations.removeElement(eventQueue);
507             }
508             updateFullRegistrations();
509         }
510         badFullRegistrations = null;
511
512     }
513
514
515     // Methods to handle persistence.
516
517     /**
```

```
518         * Allocate a persistent store manager and restore
519         * eventRegistrations and fullRegistrations.  (If they
520         * don't exist in the persistent store, instantiate
521         * new ones.) Then reconstruct eventNotifications from
522         * eventRegistrations.
523         */
524        protected void recoverRegistrations() {
525
526            Boolean enabled = new Boolean(getProperty(ENABLE_PERSISTENCE, "false"));
527            if (!enabled.booleanValue()) {
528                /*
529                 * If persistence is disabled, perform initialization,
530                 * but not recovery.
531                 */
532                System.out.println("recoverRegistrations: persistence disabled");
533                persistentStoreFile = "";
534                persistenceManager = null;
535                eventRegistrations = new Hashtable();
536                fullRegistrations = new Vector();
537                eventNotifications = new Hashtable();
538                return;
539            }
540
541                System.out.println("Persistence enabled.");
542                    persistentStoreFile = getProperty(PERSISTENCE_FILE, "");
543                    System.out.println("EventManager.persistence.FileName property set to: " + persistentStoreFile);
544
545                    /*
546                     * If no file is specified, use the default.
547                     */
548                    if (persistentStoreFile.length() == 0)
549                            persistentStoreFile = AgentConstants.AGENT_DIR + DEFAULT_PERSISTENCE_FILE;
550            System.out.println("Persistent Store File: " + persistentStoreFile);
551
552            persistenceManager = new PersistentStoreHandler(persistentStoreFile);
553
554            /*
555             * Fetch registrations from the persistent store. If they don't exist,
556             * instantiate eventRegistrations and fullRegistrations.
557             */
558            System.out.println("Fetching eventRegistrations");
559            if ((eventRegistrations =
560                (Hashtable)persistenceManager.fetchObject(EVENT_REGISTRATION_ID)) == null)
561                eventRegistrations = new Hashtable();
562
563            System.out.println("Fetching fullRegistrations");
564            if ((fullRegistrations =
```

```
565             (Vector)persistenceManager.fetchObject(FULL_REGISTRATION_ID)) == null)
566             fullRegistrations = new Vector();
567
568         /*
569          * Reconstruct the event notifications using event registrations.  Then
570          * truncate the persistent store and write out the registrations.
571          */
572         System.out.println("Reconstructing event notifications");
573         reconstructNotifications();
574
575         System.out.println("Saving registrations");
576         saveRegistrations();
577     }
578
579
580     /**
581      * Truncate persistent storage and write out event and full
582      * registrations.
583      */
584     protected void saveRegistrations() {
585         if (persistenceManager == null)
586             return;
587
588         persistenceManager.truncateFileStore();
589         persistenceManager.insertObject(EVENT_REGISTRATION_ID,
590             eventRegistrations, STORAGE_PADDING);
591         persistenceManager.insertObject(FULL_REGISTRATION_ID,
592             fullRegistrations, STORAGE_PADDING);
593     }
594
595
596     /**
597      * Update event registrations in persistent storage.
598      */
599     protected void updateEventRegistrations() {
600         if (persistenceManager == null)
601             return;
602
603         System.out.println("Updating eventRegistrations");
604         persistenceManager.updateObject(EVENT_REGISTRATION_ID, eventRegistrations);
605     }
606
607
608     /**
609      * Update full registrations in persistent storage.
610      */
611     protected void updateFullRegistrations() {
```

```
612             if (persistenceManager == null)
613                 return;
614
615             System.out.println("Updating fullRegistrations");
616             persistenceManager.updateObject(FULL_REGISTRATION_ID, fullRegistrations);
617         }
618
619
620         /**
621          * Reconstruct the event notifications table from the
622          * event registrations table.
623          */
624         protected void reconstructNotifications() {
625             eventNotifications = new Hashtable();
626
627             if (persistenceManager == null)
628                 return;
629
630                 /*
631                  * Reconstruct the eventNotifications hash table by walking
632                  * the EventRegistrations hash table and extracting registrations.
633                  */
634                 for (Enumeration registryKeys = eventRegistrations.keys();
635                     registryKeys.hasMoreElements(); ) {
636
637                     /*
638                      * For each key (event name) in the table, obtain the list of
639                      * registered event queues.
640                      */
641                     String eventName = (String)registryKeys.nextElement();
642                     System.out.println("reconstructNotifications, event = " + eventName);
643                 Vector registryList = (Vector)eventRegistrations.get(eventName);
644
645                 /*
646                  * If no objects have registered to receive this event,
647                  * remove it from the eventRegistrations hash table.
648                  */
649                 if ((registryList == null) || (registryList.isEmpty())) {
650                     System.out.println("reconstructNotifications, removing hash table entry for event " + eventNa
651                     eventRegistrations.remove(eventName);
652                     continue;
653                     }
654
655                 /*
656                  * Add the event name to the eventNotifications entry of
657                  * each of the registered event queues, creating hash table
658                  * entries, as necessary.
```

```
559                 */
560                 for (Enumeration registryElements = registryList.elements();
561                     registryElements.hasMoreElements(); ) {
562
563                     EventPost eventQueue = (EventPost)registryElements.nextElement();
564                     Vector notifyList = (Vector)eventNotifications.get(eventQueue);
565                     if (notifyList == null) {
566                         notifyList = new Vector();
567                         eventNotifications.put(eventQueue, notifyList);
568                         System.out.println("reconstructNotifications, created eventNotifications hash table entry
569                     }
570                     notifyList.addElement(eventName);
571                 }
572             }
573         }
574
575         // Misc. methods
576         /** Returns the EventManager's name */
577         protected static String getEventManagerName() {
578             return EVENT_MANAGER_NAME;
579         }
580
581
582         /** Returns the value of a property. */
583         protected String getProperty(String key, String def) {
584                     return properties.getProperty(key, def);
585             }
586
587         // Main method
588         /**
589          * EventManagerImpl bootstrap.  This is the only way to start
590          * the EventManager since it has no public constructors.
591          */
592             public static void main(String args[]) {
593                     System.setSecurityManager(new InsecurityManager());
594             try {
595                 EventManagerImpl eventManager = new EventManagerImpl();
596             } catch (Exception e) {
597                     System.out.println("An exception occurred setting up the EventManager: " +
598                                 e.getMessage());
599                     e.printStackTrace();
600             }
601             }
602
703 }
704
705
```

```
1  /*  $Header: /com/meitca/hsl/zonesjagents/event/EventManagerProxy.java 1     10/22/96 7:30p Noemi $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
4   *  Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  A client proxy for the EventManager.
11  *
12  *  $Log: /com/meitca/hsl/zonesjagents/event/EventManagerProxy.java $
13  *
14  * 1     10/22/96 7:30p Noemi
15  * EventManagerProxy is the public interface to the EventManager.  It
16  * shields users from the effects of EventManager crashes.
17  */
18
19 package com.meitca.hsl.zonesjagents.event;
20
21 import java.io.*;
22
23 import java.rmi.RemoteException;
24 import java.rmi.Naming;
25 import java.rmi.Remote;
26
27
28 /**
29  * EventManagerProxy is the public interface to the EventManager.  As a
30  * proxy for the Event Manager, it maintains a reference to the actual
31  * Event Manager and shields users from the effects of Event Manager
32  * crashes. Whenever the proxy fails to communicate with the Event Manager
33  * it attempts to re-establish a connection to it.
34  *
35  * @see     EventManager
36  * @see     EventManagerImpl
37  * @author  Noemi Paciorek
38  */
39
40 public class EventManagerProxy {
41
42     // Constants
43     protected static final int DEFAULT_RETRIES = 2;
44     protected static final long DEFAULT_TIMEOUT = 10000;
45
46
47     // Instance variables
```

WO 98/21662 PCT/US97/20232

```
48      /** Stub for EventManagerImpl's EventManager interface. */
49      protected EventManager eventManager;
50
51      /** Stub for EventManagerImpl's EventPost interface. */
52      protected EventPost eventPost;
53
54      /** The EventManagerImpl's name. */
55      protected String eventManagerName;
56
57      /**
58       * The number of times to retry when an invocation
59       * on the EventManager fails.
60       */
61      int retries;
62
63      /** The timeout period in ms. for requests to the EventManager. */
64      long timeout;
65
66
67      // Constructors
68      /**
69       * Constructs an EventManagerProxy.
70       */
71      public EventManagerProxy()
72          throws Exception {
73
74          this(DEFAULT_RETRIES, DEFAULT_TIMEOUT);
75      }
76
77
78      /**
79       * Constructs an EventManagerProxy.
80       * @param   retries The number of times to retry if an operation fails.
81       */
82      public EventManagerProxy(int retries)
83          throws Exception {
84
85          this(retries, DEFAULT_TIMEOUT);
86      }
87
88
89      /**
90       * Constructs an EventManagerProxy.
91       * @param   timeout The time to wait in ms. between retries.
92       */
93      public EventManagerProxy(long timeout)
94          throws Exception {
```

WO 98/21662 PCT/US97/20232

```
95
96          this(DEFAULT_RETRIES, timeout);
97      }
98
99
100     /**
101      * Constructs an EventManagerProxy.
102      * @param   retries The number of times to retry if an operation fails.
103      * @param   timeout The time to wait in ms. between retries.
104      */
105     public EventManagerProxy(int retries, long timeout)
106         throws Exception {
107
108         this.retries = (retries >= 0) ? retries : DEFAULT_RETRIES;
109         this.timeout = (timeout > 0) ? timeout : DEFAULT_TIMEOUT;
110
111         /*
112          * Lookup the EventManager.
113          */
114         initialize();
115     }
116
117
118     // Instance methods
119     /*
120      * Lookup the Event Manager in the RMI registry and obtain a
121      * stub for its EventManager and EventPost interfaces.
122      */
123     private void initialize()
124         throws Exception {
125
126         eventManagerName = EventManagerImpl.getEventManagerName();
127         try {
128             Remote stub = Naming.lookup(eventManagerName);
129             eventManager = (EventManager)stub;
130             eventPost = (EventPost)stub;
131         } catch (Exception e) {
132             System.out.println("initialize: error looking up EventManager: "
133                 + e.getMessage());
134             e.printStackTrace();
135             throw e;
136         }
137     }
138
139
140     /*
141      * Block for the timeout interval. Then look up the
```

```
142      * EventManager.
143      */
144     private void waitAndInitialize()
145         throws Exception {
146
147         synchronized(this) {
148             try {
149                 this.wait(timeout);
150             } catch (InterruptedException e) { }
151         }
152         initialize();
153     }
154
155     /*
156      * Convert the event class names to strings.
157      */
158     private String[] eventNametoString(EventType[] events) {
159
160         int length = events.length;
161         String eventNames[] = new String[length];
162         for (int i = 0; i < length; i++) {
163             eventNames[i] = events[i].getClass().getName();
164             System.out.println("eventNametoString, event = " + eventNames[i]);
165         }
166         return eventNames;
167     }
168
169
170
171     // Methods to handle registrations.
172
173     /**
174      * Register interest in receiving specified events.
175      * @param   events      An array of events an object is interested in receiving.
176      * @param   eventQueue  The associated event queue's EventPost interface.
177      * @exception   RemoteException If an error occurrs setting up network
178      * connections.
179      * @exception   IOException If the tranport layer generates one.
180      */
181     public void registerEvents(EventType events[], EventPost eventQueue)
182         throws Exception {
183
184         /*
185          * Convert the event class names to strings.
186          * Then invoke the EventManager's registerEvents
187          * method and retry the operation, if necessary.
188          */
```

WO 98/21662 PCT/US97/20232

```
189             String eventNames[] = eventNametoString(events);
190             for (int i = 0; i <= retries; i++) {
191                 try {
192                     eventManager.registerEvents(eventNames, eventQueue);
193                     return;
194                 } catch (RemoteException e1) {
195                     if (i == retries) {
196                         System.out.println("registerEvents: Throwing RemoteException");
197                         throw e1;
198                     }
199                     System.out.println("registerEvents: Caught RemoteException");
200                 } catch (IOException e2) {
201                     if (i == retries) {
202                         System.out.println("registerEvents: Throwing IOException");
203                         throw e2;
204                     }
205                     System.out.println("registerEvents: Caught IOException");
206                 }
207                 /*
208                  * Block for the timeout interval. Then look up the
209                  * EventManager, before retrying the operation.
210                  */
211                 waitAndInitialize();
212             }
213         }
214
215
216         /**
217          * Register interest in receiving all events posted to this EventManager.
218          * @param   eventQueue  The associated event queue's EventPost interface.
219          * @exception  RemoteException If an error occurrs setting up network
220          * connections.
221          * @exception   IOException If the tranport layer generates one.
222          */
223         public void registerAll(EventPost eventQueue)
224             throws Exception {
225
226             /*
227              * Invoke the EventManager's registerAll
228              * method and retry the operation, if necessary.
229              */
230             for (int i = 0; i <= retries; i++) {
231                 try {
232                     eventManager.registerAll(eventQueue);
233                     return;
234                 } catch (RemoteException e1) {
235                     if (i == retries) {
```

```
                    System.out.println("registerAll: Throwing RemoteException");
                    throw e1;
                }
                System.out.println("registerAll: Caught RemoteException");
            } catch (IOException e2) {
                if (i == retries) {
                    System.out.println("registerAll: Throwing IOException");
                    throw e2;
                }
                System.out.println("registerAll: Caught IOException");
            }
            /*
             * Block for the timeout interval. Then look up the
             * EventManager, before retrying the operation.
             */
            waitAndInitialize();
        }
    }


    // Methods to unregister events.
    /**
     * Remove registration for specified events.
     * @param   events      An array of events an object is interested in receiving.
     * @param   eventQueue  The associated event queue's EventPost interface.
     * @exception   RemoteException If an error occurrs setting up network
     * connections.
     * @exception   EventManagerException If the event queue was not registered
     * for one or more of the specified exceptions.
     * @exception   IOException If the tranport layer generates one.
     */
    public void unregisterEvents(EventType events[], EventPost eventQueue)
        throws Exception {

        /*
         * Convert the event class names to strings.
         * Then invoke the EventManager's registerEvents
         * method and retry the operation, if necessary.
         */
        String eventNames[] = eventNametoString(events);
        for (int i = 0; i <= retries; i++) {
            try {
                eventManager.unregisterEvents(eventNames, eventQueue);
                return;
            } catch (RemoteException e1) {
                if (i == retries) {
                    System.out.println("unregisterEvents: Throwing RemoteException");
```

```
283                     throw e1;
284                 }
285                 System.out.println("unregisterEvents: Caught RemoteException");
286             } catch (IOException e2) {
287                 if (i == retries) {
288                 System.out.println("unregisterEvents: Throwing IOException");
289                     throw e2;
290                 }
291                 System.out.println("unregisterEvents: Caught IOException");
292             }
293             /*
294              * Block for the timeout interval. Then look up the
295              * EventManager, before retrying the operation.
296              */
297             waitAndInitialize();
298         }
299     }
300
301
302     /**
303      * Remove registration for all events.
304      * @param   eventQueue  The associated event queue's EventPost interface.
305      * This method is often called by finalizers, so it does not
306      * propagate any errors.
307      */
308     public void unregisterAll(EventPost eventQueue) {
309         /*
310          * Invoke the EventManager's registerAll
311          * method and retry the operation, if necessary.
312          */
313         for (int i = 0; i <= retries; i++) {
314             try {
315                 eventManager.unregisterAll(eventQueue);
316                 return;
317             } catch (RemoteException e1) {
318                 System.out.println("unregisterAll: Caught RemoteException");
319             } catch (IOException e2) {
320                 System.out.println("unregisterAll: Caught IOException");
321             }
322             /*
323              * Block for the timeout interval. Then look up the
324              * EventManager, before retrying the operation.
325              */
326             try {
327                 waitAndInitialize();
328             } catch (Exception e3) {
329                 System.out.println("waitAndInitialize failed");
```

WO 98/21662 PCT/US97/20232

```
330             }
331         }
332     }
333
334
335     // Methods to post events.
336
337     /**
338      * Event notification interface.
339      * @param   event   The event to post.
340      * @exception   RemoteException If an error occurrs setting up network
341      * connections.
342      * @exception   EventException  If an error occurs posting the event.
343      * @exception   IOException If the tranport layer generates one.
344      */
345     public void postEvent(EventType event)
346         throws Exception {
347         /*
348          * Invoke the EventManager's eventPost
349          * method and retry the operation, if necessary.
350          */
351         for (int i = 0; i <= retries; i++) {
352             try {
353                 eventPost.postEvent(event);
354                 return;
355             } catch (RemoteException e1) {
356                 if (i == retries) {
357                     System.out.println("postEvent: Throwing RemoteException");
358                     throw e1;
359                 }
360                 System.out.println("postEvent: Caught RemoteException");
361             } catch (IOException e2) {
362                 if (i == retries) {
363                     System.out.println("postEvent: Throwing IOException");
364                     throw e2;
365                 }
366                 System.out.println("postEvent: Caught IOException");
367             }
368             /*
369              * Block for the timeout interval. Then look up the
370              * EventManager, before retrying the operation.
371              */
372             waitAndInitialize();
373         }
374     }
375 }
376
```

EventManagerProxy.java_1 Mon Nov 11 16:26:42 1996 9

377

```
 1  /*  $Header: /com/meitca/hsl/zonesjagents/event/EventPost.java 2     10/22/96 7:10p Noemi $
 2   *
 3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
 4   *  Information Technology Center America.
 5   *      All rights reserved.
 6   *
 7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 8   *
 9   *  DESCRIPTION
10   *  Interface for event notification
11   *
12   *  $Log: /com/meitca/hsl/zonesjagents/event/EventPost.java $
13   *
14   * 2     10/22/96 7:10p Noemi
15   * Changed the throws clause of the postEvent method.
16   *
17   * 1     10/14/96 6:57p Noemi
18   * Interface used for posting events.
19   */
20
21  package com.meitca.hsl.zonesjagents.event;
22
23  import java.io.*;
24
25  import java.rmi.Remote;
26  import java.rmi.RemoteException;
27
28  /**
29   * EventPost is a distributed RMI interface for notification of events.
30   *
31   * @author Noemi Paciorek
32   */
33
34  public interface EventPost
35      extends Remote {
36
37      // Instance methods.
38      /**
39       * Event notification interface.
40       * @param   event   The event to post.
41       * @exception   RemoteException If an error occurrs setting up network
42       * connections.
43       * @exception   EventException  If an error occurs posting the event.
44       */
45      public void postEvent(EventType event)
46          throws RemoteException, EventException, IOException;
47
```

EventPost.java_1 Mon Nov 11 16:25:42 1996 2

```
48  }
```

```
1  /*  $Header: /com/meitca/hsl/zonesjagents/event/EventQueueImpl.java 1      10/14/96 6:58p Noemi $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
4   *  Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  Event queue distributed object.
11  *
12  *  $Log: /com/meitca/hsl/zonesjagents/event/EventQueueImpl.java $
13  *
14  * 1     10/14/96 6:58p Noemi
15  * Event queue implementation.
16  */
17
18 package com.meitca.hsl.zonesjagents.event;
19
20 import java.util.*;
21
22 import java.rmi.server.UnicastRemoteServer;
23 import java.rmi.RemoteException;
24 import java.rmi.Naming;
25 import java.rmi.NotBoundException;
26
27 /**
28  * EventQueueImpl is an event queue that implements the RMI
29  * EventPost interface.
30  *
31  * Event notification is performed by appending an event to the queue.
32  * Each queue has an associated EventQueueThread started by the constructor.
33  * The thread dequeues events and calls their handlers.
34  *
35  * @see     EventQueueThread
36  * @see     EventPost
37  * @author  Noemi Paciorek
38  */
39
40 public class EventQueueImpl
41     extends UnicastRemoteServer
42     implements EventPost {
43
44     // Instance variables
45     /** The event queue */
46     protected Vector eventQueue;
47
```

WO 98/21662 PCT/US97/20232

```
48      /** The thread that manages this queue */
49      protected EventQueueThread queueThread;
50
51      // Constructors
52      /**
53       * Protected Default Constructor for EventQueueImpl.
54       * @exception   IllegalAccessException  Whenever called.
55       */
56      protected EventQueueImpl()
57          throws RemoteException, IllegalAccessException {
58
59          System.out.println("Default constructor called for EventQueueImpl");
60          throw new IllegalAccessException("EventQueueImpl default constructor");
61      }
62
63      /**
64       * Constructs an EventQueueImpl
65       * @param   handler An EventHandler interface.
66       */
67      public EventQueueImpl(EventHandler handler)
68          throws RemoteException {
69          eventQueue = new Vector();
70          queueThread = new EventQueueThread(this, handler);
71      }
72
73      /**
74       * Constructs an EventQueueImpl
75       * @param   handler An EventHandler interface.
76       * @param   timeout The time in ms to wait between consecutive queue
77       * examinations.
78       */
79      public EventQueueImpl(EventHandler handler, long timeout)
80          throws RemoteException {
81          eventQueue = new Vector();
82          queueThread = new EventQueueThread(this, handler, timeout);
83      }
84
85      // Instance methods
86      /**
87       * Posts an event by enqueueing it and waking up the
88       * queue's thread.
89       * @param   event   The event to post.
90       */
91      public synchronized void postEvent(EventType event) {
92          System.out.println("EventQueueImpl, posting event");
93          enqueue(event);
94          this.notifyAll();
```

```
    }

    /**
     * Enqueues an event.
     * @param   event   The event to queue.
     */
    protected final synchronized void enqueue(EventType event) {
        eventQueue.addElement(event);
    }

    /**
     * Dequeues an event.
     * @return   The first event in the queue.
     */
    protected final synchronized EventType dequeue()
        throws ArrayIndexOutOfBoundsException {

        if (!eventQueue.isEmpty()) {
            EventType event = (EventType)(eventQueue.firstElement());
            eventQueue.removeElementAt(0);
            return event;
        }
        return null;
    }
}
```

```
1  /*  $Header: /com/meitca/hsl/zonesjagents/event/EventQueueThread.java 1     10/14/96 6:59p Noemi $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
4   *  Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  Thread that manages an event queue.
11  *
12  *  $Log: /com/meitca/hsl/zonesjagents/event/EventQueueThread.java $
13  *
14  * 1     10/14/96 6:59p Noemi
15  * Thread to manage the event queue.
16  */
17
18 package com.meitca.hsl.zonesjagents.event;
19
20 /**
21  * EventQueueThread is a Thread that manages an event queue.
22  * Event notification is implemented by appending an event to the queue
23  * and waking up its EventQueueThread to process the event by calling its
24  * event handler.
25  * The EventQueueThread is instantiated by the EventQueueImpl constructor.
26  *
27  * @see     EventQueueImpl
28  * @see     EventPost
29  * @see     EventHandler
30  * @author  Noemi Paciorek
31  */
32 class EventQueueThread
33    extends Thread {
34
35    // Constants
36    /** The default timeout value in ms. */
37    public static final long DEFAULT_TIMEOUT = 1000;
38
39    // Instance variables
40    /** This thread's event queue */
41    protected EventQueueImpl eventQueue;
42
43    /** The application's event handler */
44    protected EventHandler eventHandler;
45
46    /** The time in ms between consecutive examinations of the queue. */
47    protected long timeout;
```

WO 98/21662 PCT/US97/20232

```
48
49      // Constructors
50      /**
51       * Protected Default Constructor for an EventQueueThread.
52       * @exception   IllegalAccessException  Whenever called.
53       */
54      protected EventQueueThread(.
55          throws IllegalAccessException {
56
57          System.out.println("Default constructor called for EventQueueThread");
58          throw new IllegalAccessException("EventQueueThread default constructor");
59      }
60
61      /** Constructs an EventQueueThread.
62       * @param   queue   The associated event queue.
63       * @param   handler The associated event handler interface.
64       */
65      public EventQueueThread(EventQueueImpl queue, EventHandler handler) {
66          this(queue, handler, DEFAULT_TIMEOUT);
67      }
68
69      /** Constructs an EventQueueThread.
70       * @param   queue   The associated event queue.
71       * @param   handler The associated event handler interface.
72       * @param   timeout The time in ms to wait between consecutive queue
73       * examinations.
74       */
75      public EventQueueThread(EventQueueImpl queue, EventHandler handler, long timeout) {
76          eventQueue = queue;
77          eventHandler = handler;
78          this.timeout = timeout;
79          this.start();
80      }
81
82      // Instance methods
83
84      /**
85       * The thread's run method.
86       */
87      public void run() {
88          while (true) {
89              try {
90                  synchronized(eventQueue) {
91                      eventQueue.wait(timeout);
92                  }
93              } catch (InterruptedException e) { }
94
```

EventQueueThread.java_1 Mon Nov 11 16:26:42 1996 3

```
 95          EventType event;
 96          while ((event = eventQueue.dequeue()) != null) {
 97            System.out.println("EventQueueThread, handling event");
 98            try {
 99              eventHandler.handleEvent(event);
100              } catch (Exception ex) {
101                System.out.println("EventQueueThread: can't handle event: " +
102                    ex.getMessage());
103                ex.printStackTrace();
104
105            }
106          }
107        }
108
109
110
111
112
113
114
115
116
```

```
 1  /*  $Header: /ccm/meitca/hsl/zonesjagents/event/EventType.java 1     10/14/96 6:55p Noemi $
 2   *
 3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
 4   *  Information Technology Center America.
 5   *      All rights reserved.
 6   *
 7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
 8   *
 9   *  DESCRIPTION
10   *  Base class for distributed events.
11   *
12   *  $Log: /com/meitca/hsl/zonesjagents/event/EventType.java $
13   *
14   * 1     10/14/96 6:55p Noemi
15   * Base class for distributed events.
16   */
17
18  package com.meitca.hsl.zonesjagents.event;
19
20  import com.meitca.hsl.util.UniqueID;
21
22  /**
23   * Base class for distributed events.  EventType describes an event.
24   * It contains: a unique ID (generated by the EventManager) and a description
25   * (optionally passed to the constructors).
26   *
27   * Applications must derive and handle their own event types.
28   *
29   * @author Noemi Paciorek
30   */
31
32  abstract public class EventType {
33
34      // Constants
35      private static final String UNKNOWN = "Unknown";
36      private static final String TYPE = "event";
37
38      // Instance variables
39      /** A unique ID representing the event */
40      protected String eventID;
41
42      /** A description of the event */
43      protected String description;
44
45
46          // Constructors
47          /** Constructs an EventType */
```

```
48          public EventType() {
49              this(UNKNOWN);
50          }
51
52          /** Constructs an EventType
53           * @param   description A String describing the event.
54           */
55          public EventType(String description) {
56              eventID = UniqueID.getNextID(TYPE, this);
57              System.out.println("eventID = " + eventID);
58              this.description = description;
59          }
60
61
62          // Instance methods
63          /**
64           * String representation of event.
65           * @return A String representation of the event.
66           */
67          public final String toString() {
68              return (eventID + "/" + description);
69          }
70
71          /**
72           * Retrieve an event's ID.
73           * @return  The event's ID
74           */
75      public final String getEventID() {
76          return eventID;
77      }
78
79      /**
80       * Retrieve an event's description.
81       * @return  A description of the event.
82       */
83      public final String getEventDescription() {
84          return description;
85      }
86
87  }
88
89
90
91
92
93
94
```

```
EventType.java_1      Mon Nov 11 16:25:43 1996      3

   95
   96
   97
   98
```

```
1  /*  $Header: /com/meitca/hsl/zonesjagents/shared/PersistentStoreHandler.java 1     10/22/96 7:32p Noemi $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
4   *  Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  Persistent Store Manager wrapper that handles exceptions.
11  *
12  *  $Log: /com/meitca/hsl/zonesjagents/shared/PersistentStoreHandler.java $
13  *
14  * 1     10/22/96 7:32p Noemi
15  * A generic wrapper for PersistentStoreManager that catches exceptions
16  * and retries operations, as necessary.
17  */
18
19 package com.meitca.hsl.zonesjagents.shared;
20
21 import com.meitca.hsl.zonesjagents.persistence.*;
22
23 /**
24  * A wrapper class for the persistent storage manager
25  * (persistence.PersistentStoreManager)that catches
26  * all its exceptions and performs the appropriate cleanup.
27  * @see     PersistentStoreManager
28  * @author  Noemi Paciorek
29  */
30 public class PersistentStoreHandler {
31
32     // Instance variables
33     /** A reference to the persistent store manager. */
34     protected PersistentStoreManager persistenceManager;
35
36     // Constructors
37     /**
38      * Constructs a PersistentStoreHandler.
39      * @param filename  The name of the persistent storage file.
40      */
41     public PersistentStoreHandler(String filename) {
42         try {
43             persistenceManager = new PersistentStoreManager(filename);
44         } catch (PersistentStoreException e) {
45             System.out.println("PersistentStoreHandler: can't create persistent store manager " + e.getMessa
46             e.printStackTrace();
47             persistenceManager = null;
```

```
48          }
49      }
50
51      // Finalizer
52      /**
53       * Deletes reference to PersistentStoreManager.
54       */
55      protected void finalize()
56          throws Throwable {
57
58          System.out.println("PersistentStoreHandler Finalizer called");
59
60          // Delete reference to Persistent Store Manager.
61          persistenceManager = null;
62
63          super.finalize();
64      }
65
66
67
68      // Instance methods.
69      /**
70       * Find an object in the persist store.
71       * @param   object  The Object to locate.
72       * @return          The Object's persistent storage ID.
73       */
74      public int findObject(Object object) {
75          return persistenceManager.findObject(object);
76      }
77
78      /**
79       * Insert an Object into persistent storage.
80       * If a storage manager exception occurs, attempt to
81       * truncate the persistent storage and disable persistence.
82       * @param   object  The Object represented by id.
83       * @return          The Object's persistent storage ID.
84       * A return value of PersistentStoreManager. INVALID_OID indicates
85       * an unsuccessful attempt to insert the object.
86       */
87      public int insertObject(Object object) {
88          return insertObject(0, object, 0);
89      }
90
91
92      /**
93       * Insert an Object into persistent storage.
94       * If a storage manager exception occurs, attempt to
```

```
95      * truncate the persistent storage and disable persistence.
96      * @param   Object  The Object represented by id.
97      * @param   space   The additional space in bytes to be preallocated
98      * for this object.
99      * @return          The Object's persistent storage ID.
100     * A return value of PersistentStoreManager. INVALID_OID indicates
101     * an unsuccessful attempt to insert the object.
102     */
103    public int insertObject(Object object, int space) {
104        return insertObject(0, object, space);
105    }
106
107
108    /**
109     * Insert an Object into persistent storage.
110     * If a storage manager exception occurs, attempt to
111     * truncate the persistent storage and disable persistence.
112     * @param   id      The Object's persistent storage ID.
113     * @param   Object  The Object represented by id.
114     * @return          The Object's persistent storage ID.
115     * A return value of PersistentStoreManager. INVALID_OID indicates
116     * an unsuccessful attempt to insert the object.
117     */
118    public int insertObject(int id, Object object) {
119        return insertObject(id, object, 0);
120    }
121
122
123    /**
124     * Insert an Object into persistent storage.
125     * If a storage manager exception occurs, attempt to
126     * truncate the persistent storage and disable persistence.
127     * @param   id      The Object's persistent storage ID.
128     * @param   Object  The Object represented by id.
129     * @param   space   The additional space in bytes to be preallocated
130     * for this object.
131     * A return value of PersistentStoreManager. INVALID_OID indicates
132     * an unsuccessful attempt to insert the object.
133     */
134    public int insertObject(int id, Object object, int space) {
135        if (persistenceManager == null)
136            return PersistentStoreManager.INVALID_OID;
137
138        try {
139            if (id == 0)
140                return persistenceManager.insertObject(object, space);
141            return persistenceManager.insertObject(id, object, space);
```

WO 98/21662 PCT/US97/20232

```
142         } catch (PersistentStoreDuplicateObjectException e1) {
143             try {
144                 id = persistenceManager.findObject(object);
145                 if (id != PersistentStoreManager.INVALID_OID) {
146                     persistenceManager.updateObject(id, object);
147                     return id;
148                 }
149             } catch (PersistentStoreException e2) {
150                 System.out.println("insertObject: duplicate object found" + e2.getMessage());
151                 e2.printStackTrace();
152                 truncateFileStore();
153                 persistenceManager = null;
154             }
155         } catch (PersistentStoreInvalidObjectException e3) {
156                 System.out.println("insertObject: invalid ID" + e3.getMessage());
157                 e3.printStackTrace();
158         } catch (PersistentStoreException e4) {
159             System.out.println("insertObject: can't insert object " + e4.getMessage());
160             e4.printStackTrace();
161             truncateFileStore();
162             persistenceManager = null;
163         }
164         return PersistentStoreManager.INVALID_OID;
165     }
166
167
168     /**
169      * Fetch an Object from persistent storage.
170      * If a storage manager exception occurs, attempt to
171      * truncate the persistent storage and disable persistence.
172      * @param   id  The Object's persistent storage ID.
173      * @return  Object  The Object represented by id, or null,
174      * if the Object was not found in persistent storage.
175      */
176     public Object fetchObject(int id) {
177         if (persistenceManager == null)
178             return null;
179
180         try {
181             return persistenceManager.fetchObject(id);
182         } catch (PersistentStoreObjectNotFoundException e1) {
183                 System.out.println("fetchObject: Object not found in persistent store, ID = " + id);
184         } catch (PersistentStoreInvalidObjectException e2) {
185                 System.out.println("fetchObject: invalid ID" + e2.getMessage());
186                 e2.printStackTrace();
187         } catch (PersistentStoreException e3) {
188             System.out.println("fetchObject: can't fetch object " + e3.getMessage());
```

```
            e3.printStackTrace();
            truncateFileStore();
            persistenceManager = null;
        }
        return null;
    }


    /**
     * Update an Object in persistent storage.
     * If a storage manager exception occurs, attempt to
     * truncate the persistent storage and disable persistence.
     * @param   id      The Object's persistent storage ID.
     * @param  Object   The updated Object represented by id.
     * @return          A boolean indicating if the update was successful.
     */
    public boolean updateObject(int id, Object object) {
        if (persistenceManager == null)
            return false;

        try {
            persistenceManager.updateObject(id, object);
            return true;
        } catch (PersistentStoreObjectNotFoundException e1) {
            System.out.println("updateObject: Object not found in persistent store, ID = " + id);
            try {
                persistenceManager.insertObject(id, object);
                return true;
            } catch (PersistentStoreException e2) {
                System.out.println("updateObject: can't insert object " + e2.getMessage());
                e2.printStackTrace();
                truncateFileStore();
                persistenceManager = null;
            }
        } catch (PersistentStoreInvalidObjectException e3) {
                System.out.println("updateObject: invalid ID" + e3.getMessage());
                e3.printStackTrace();
        } catch (PersistentStoreException e4) {
            System.out.println("updateObject: can't update object " + e4.getMessage());
            e4.printStackTrace();
            truncateFileStore();
            persistenceManager = null;
        }
        return false;
    }
```

```
283             try {
284                 persistenceManager.truncateFileStore();
285                 return true;
286             } catch(Exception e) {
287                 System.out.println("truncateFileStore: can't truncate store " + e.getMessage());
288                 e.printStackTrace();
289                 persistenceManager = null;
290                 return false;
291             }
292         }
293
294         /**
295          * Return a reference to the persistent storage manager.
296          */
297         public PersistentStoreManager getPersistenceManager() {
298             return persistenceManager;
299         }
300     }
301
302
303
304
```

```
236     /**
237      * Delete an object from persistent storage.
238      * If a storage manager exception occurs, attempt to
239      * truncate the persistent storage and disable persistence.
240      * @param   id  The Object's persistent storage ID.
241      * @return      A boolean indicating if the deletion was successful.
242      */
243     public boolean deleteObject(int id) {
244         if (persistenceManager == null)
245             return false;
246
247         try {
248             persistenceManager.deleteObject(id);
249             return true;
250         } catch (PersistentStoreObjectNotFoundException e1) {
251                 System.out.println("deleteObject: Object not found in persistent store, ID = " + id);
252         } catch (PersistentStoreInvalidObjectException e2) {
253                 System.out.println("deleteObject: invalid ID" + e2.getMessage());
254                 e2.printStackTrace();
255         } catch (PersistentStoreException e3) {
256             System.out.println("deleteObject: can't delete object " + e3.getMessage());
257             e3.printStackTrace();
258             truncateFileStore();
259             persistenceManager = null;
260         }
261         return false;
262     }
263
264
265     /**
266      * Disable storage of objects in persistent store.  This method
267      * also truncates the persistent storage file to 0 bytes.
268      */
269     public void disablePersistence() {
270         truncateFileStore();
271         persistenceManager = null;
272     }
273
274
275     /**
276      * Truncate the persistent storage file to 0 bytes. If a storage manager
277      * exception occurs, disable persistence.
278      */
279     public boolean truncateFileStore() {
280         if (persistenceManager == null)
281             return false;
282
```

```
1  /*  $Header: /com/meitca/hsl/util/UniqueID.java 1     10/15/96 7:20a Billp $
2   *
3   *  Copyright  1996 Horizon Systems Laboratory, Mitsubishi Electric
4   *  Information Technology Center America.
5   *      All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  This class generates IDs that are unique across distributed systems.
11  *  $Log: /com/meitca/hsl/util/UniqueID.java $
12  *
13  * 1     10/15/96 7:20a Billp
14  */
15
16 package com.meitca.hsl.util;
17
18 import java.util.*;
19
20 import java.net.InetAddress;
21
22 /**
23  * This class generates IDs that are unique within a Java VM.
24  * It is used by classes that need to generate unique IDs.
25  *
26  * @author Noemi Paciorek
27  */
28 public class UniqueID {
29
30     // Class variables
31     /** A counter used in generating IDs. */
32     protected static long IDNum = 0;
33
34     /** A unique ID representing the host */
35     protected static String host;
36
37         // Static initializer
38         static {
39             /*
40              * Try to obtain the host's name and IP address.  If this fails,
41              * construct a pseudo-random name and hope for the best!  (Of course,
42              * this should never happen.)
43              */
44             try {
45                 host = InetAddress.getLocalHost().toString();
46             } catch (Exception e) {
47                 host = "__HOSTID__" + new Random().nextLong();
```

WO 98/21662 PCT/US97/20232

```
48              }
49      }
50
51
52      // Constructors
53      // No accessible constructors.
54      /**
55       * Protected Default Constructor for UniqueID.
56       * This class never needs to be instantiated since all its variables
57       * and methods are static.
58       * @exception   IllegalAccessException  Whenever called.
59       */
60      protected UniqueID()
61          throws IllegalAccessException {
62
63          System.out.println("Default constructor called for UniqueID");
64          throw new IllegalAccessException("UniqueID default constructor");
65      }
66
67
68          // Class methods
69      /**
70       * This method attempts to create an ID that is unique across
71       * Java VMs.  The ID is created from: the hostname, an application type
72       * (e.g., agent), the current time in ms, and the virtual memory
73       * address of the object being created.
74       *
75       * @param   type    A String containing an application type, e.g., "agent".
76       * @param   object  The "this" reference of the calling object.
77       */
78      public static String getNextID(String type, Object object) {
79              return (host + "/" + type + nextIDNum() + "/" +
80                  System.currentTimeMillis() + Integer.toHexString(object.hashCode()));
81      }
82
83
84          private static final synchronized long nextIDNum() {
85              return ++IDNum;
86          }
87
88  }
```

CLAIMS

What is claimed is:

1. In a computer network including at least a first computer and a second computer, an apparatus for performing a task in a second computer, comprising:

a routine that generates a mobile agent object in the first computer, the mobile agent including both data and executable code;

a protocol for transmission of the mobile agent from the first computer to the second computer, wherein execution of the mobile agent in the second computer following such transmission prompts the task to be performed in the second computer; and

an itinerary containing a reference to each destination computer to which the mobile agent is designated to migrate and a reference to a method associated with each respective destination computer reference, the respective method being invoked when the mobile agent is present on the respective destination computer.

2. The apparatus of claim 1 wherein the transmission protocol includes a routine in the first computer for serializing the mobile agent to generate a stream of data therefrom.

3. The apparatus of claim 2 wherein the transmission protocol includes a routine in the second computer for deserializing the stream of bytes to regenerate the mobile agent therefrom.

4. The apparatus of claim 1 wherein the task is data gathering, and wherein the mobile agent is transmitted back to the first computer from the second computer following completion of such data gathering.

5. The apparatus of claim 1 wherein the mobile agent

migrates to a plurality of destination computers within the computer network.

6. The apparatus of claim 1 wherein at least one destination computer reference is a Uniform Resource Locator.

7. The apparatus of claim 1 wherein at least a portion of the executable code required for invoking the method is retrieved from a mobile codebase portion of the executable code included in the mobile agent.

8. The apparatus of claim 7 wherein the mobile agent includes a reference to a home codebase located on the first computer, the reference being employed to retrieve required executable code which is not present in the mobile codebase.

9. The apparatus of claim 8 wherein required executable code is sought first in the computer on which the mobile agent is executing, then in the mobile codebase, and finally on the home codebase.

10. The apparatus of claim 8 wherein a new thread is formed for execution of the mobile agent in the destination computer following receipt of the mobile agent, and wherein a security feature prevents the mobile agent from operating outside a scope assigned to such thread.

11. The apparatus of claim 8 wherein the mobile agent further includes sub-objects.

12. The apparatus of claim 1 wherein a persistent local copy of the mobile agent is stored following receipt of the mobile agent, and wherein such copy is overwritten with an updated persistent copy following agent execution.

13. In a computer network including at least a first computer and a second computer, a method for performing a task in a second computer, comprising the steps of:

generating a mobile agent object in the first computer, the mobile agent including both data and executable code;

transmitting the mobile agent from the first computer to the second computer;

receiving the mobile agent in the second computer; and

executing the mobile agent in the second computer in accordance with an itinerary which contains a reference to each destination computer to which the mobile agent is designated to migrate and a reference to a method associated with each respective destination computer reference, the method associated with the second computer being invoked when the mobile agent is present on the second computer.

14. The method of claim 13 including the further step of serializing the mobile agent to generate a stream of data therefrom in the first computer.

15. The method of claim 14 including the further step of deserializing the stream of bytes to regenerate the mobile agent therefrom in the second computer.

16. The method of claim 13 wherein the task is data gathering, and including the further step of transmitting the mobile agent back to the first computer from the second computer following completion of such data gathering.

17. The method of claim 13 including the further step of causing the mobile agent to sequentially migrate to a plurality of destination computers within the computer network.

18. The method of claim 13 including the further step of employing a Uniform Resource Locator as one of the at least one destination computer reference.

19. The method of claim 13 including the further step of retrieving at least a portion of the executable code required for invoking the method from a mobile codebase portion of the executable code included in the mobile agent.

20. The method of claim 19 wherein the mobile agent includes a reference to a home codebase located on the first computer, and including the further step of employing the reference to retrieve required executable code which is not present in the mobile codebase.

21. The method of claim 20 including the further step of searching for required executable code first in the computer on which the mobile agent is executing, then in the mobile codebase, and finally on the home codebase.

22. The method of claim 21 including the further step of forming a new thread in the destination computer for execution of the mobile agent following receipt thereof, and wherein a security feature prevents the mobile agent from operating outside a scope assigned to such thread.

23. The method of claim 21 including the further step of inserting sub-objects into the mobile agent.

24. The method of claim 13 including the further step of storing a persistent local copy of the mobile agent following receipt thereof, and overwriting such copy with an updated persistent copy following agent execution.

25. In a computer network including at least a first computer and a second computer, an apparatus for performing a task in a second computer, comprising:

a routine that generates a mobile agent object in the first computer, the mobile agent including both data and executable code;

a protocol for transmission of the mobile agent from the first computer to the second computer, wherein execution of

the mobile agent in the second computer following such transmission prompts the task to be performed in the second computer, at least a portion of the executable code required for execution being retrieved from a mobile codebase portion of the executable code included in the mobile agent; and

a reference to a home codebase located on the first computer, the reference being employed to retrieve required executable code which is not present in the mobile codebase.

26. The apparatus of claim 25 wherein the transmission protocol includes a routine in the first computer for serializing the mobile agent to generate a stream of data therefrom.

27. The apparatus of claim 26 wherein the transmission protocol includes a routine in the second computer for deserializing the stream of bytes to regenerate the mobile agent therefrom.

28. The apparatus of claim 25 wherein the task is data gathering, and wherein the mobile agent is transmitted back to the first computer from the second computer following completion of such data gathering.

29. The apparatus of claim 25 wherein the mobile agent migrates to a plurality of destination computers within the computer network.

30. The apparatus of claim 29 wherein the mobile agent further includes an itinerary containing a reference to each destination computer to which the agent is designated to migrate and a reference to a method associated with each respective destination computer reference, the respective method being invoked when the mobile agent is present on the respective destination computer.

31. The apparatus of claim 30 wherein at least one destination computer reference is a Uniform Resource Locator.

32. The apparatus of claim 25 wherein required executable code is sought first in the computer on which the mobile agent is executing, then in the mobile codebase, and finally on the home codebase.

33. The apparatus of claim 25 wherein a new thread is formed for execution of the mobile agent in the destination computer following receipt of the mobile agent, and wherein a security feature prevents the mobile agent from operating outside a scope assigned to such thread.

34. The apparatus of claim 25 wherein the mobile agent further includes sub-objects.

35. The apparatus of claim 30 wherein a persistent local copy of the mobile agent is stored following receipt of the mobile agent, and wherein such copy is overwritten with an updated persistent copy following agent execution.

36. In a computer network including at least a first computer and a second computer, a method for performing a task in a second computer, comprising the steps of:

generating a mobile agent object in the first computer, the mobile agent including both data and executable code;

transmitting the mobile agent from the first computer to the second computer;

receiving the mobile agent in the second computer;

retrieving at least a portion of the executable code required for invoking the method from a mobile codebase portion of the executable code included in the mobile agent;

employing a reference to a home codebase located on the first computer to retrieve required executable code which is not present in the mobile codebase; and

executing the mobile agent in the second computer such that the mobile agent prompts the task to be performed in the second computer.

37. The method of claim 36 including the further step of serializing the mobile agent to generate a stream of data therefrom in the first computer.

38. The method of claim 37 including the further step of deserializing the stream of bytes to regenerate the mobile agent therefrom in the second computer.

39. The method of claim 36 wherein the task is data gathering, and including the further step of transmitting the mobile agent back to the first computer from the second computer following completion of such data gathering.

40. The method of claim 36 including the further step of causing the mobile agent to sequentially migrate to a plurality of destination computers within the computer network.

41. The method of claim 40 wherein the mobile agent further includes an itinerary which contains a reference to each destination computer to which the agent is designated to migrate and a reference to a method associated with each respective destination computer reference, and including the further step of invoking the method associated with the respective destination computer when the mobile agent is present on the respective destination computer.

42. The method of claim 41 including the further step of employing a Uniform Resource Locator as one of the at least one destination computer reference.

43. The method of claim 36 including the further step of searching for required executable code first in the computer on which the mobile agent is executing, then in the mobile codebase, and finally on the home codebase.

44. The method of claim 36 including the further step of forming a new thread in the destination computer for

execution of the mobile agent following receipt thereof, and wherein a security feature prevents the mobile agent from operating outside a scope assigned to such thread.

45. The method of claim 36 including the further step of inserting sub-objects into the mobile agent.

46. The method of claim 41 including the further step of storing a persistent local copy of the mobile agent following receipt thereof, and overwriting such copy with an updated persistent copy following agent execution.

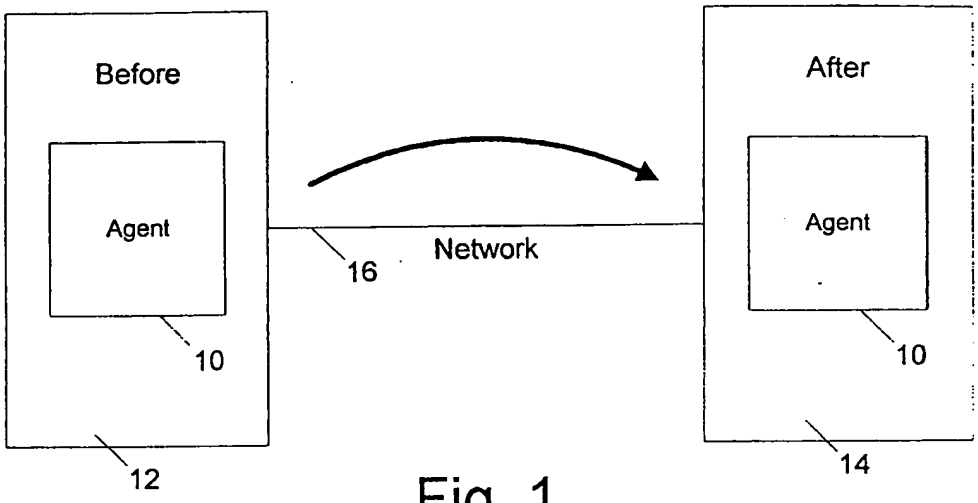

Before
Agent
10
12
16
Network
After
Agent
10
14

Fig. 1

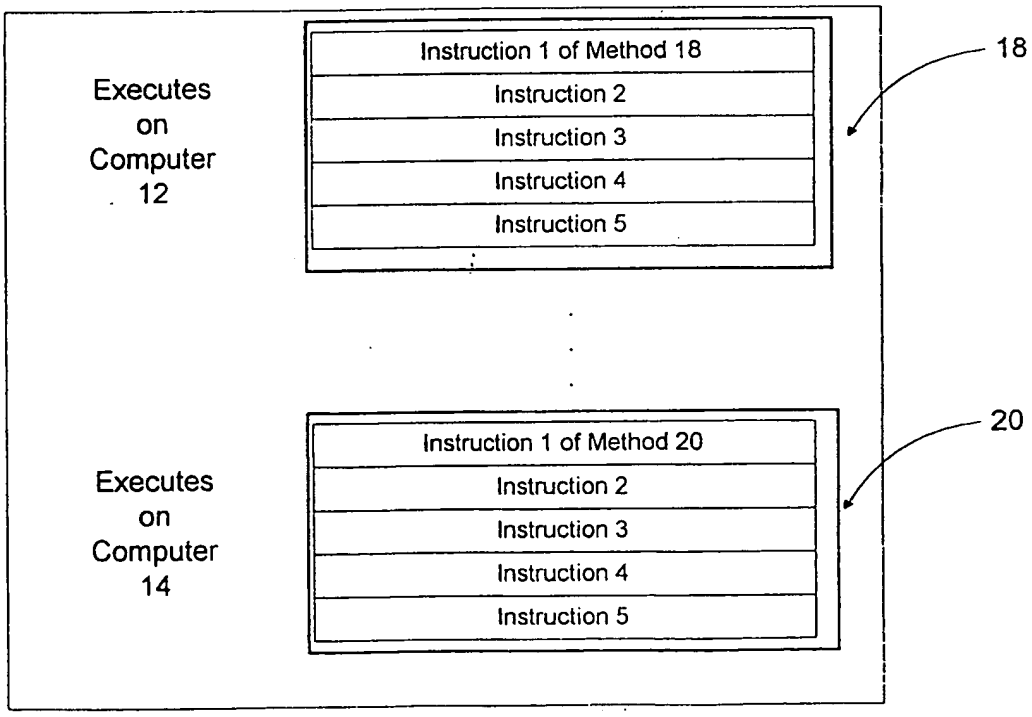

Executes on Computer 12
Instruction 1 of Method 18
Instruction 2
Instruction 3
Instruction 4
Instruction 5
18
Executes on Computer 14
Instruction 1 of Method 20
Instruction 2
Instruction 3
Instruction 4
Instruction 5
20

Fig. 2

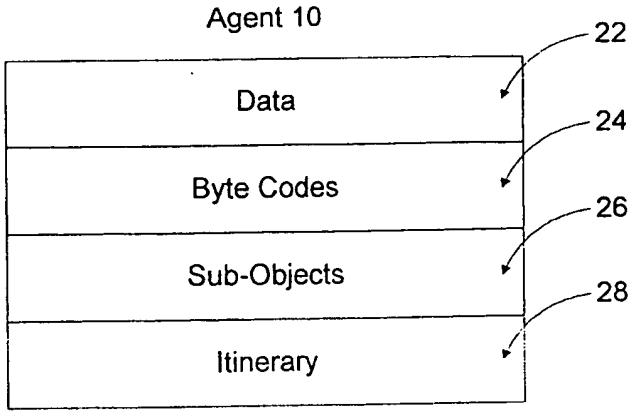

Agent 10
Data
Byte Codes
Sub-Objects
Itinerary
22
24
26
28

Fig. 3

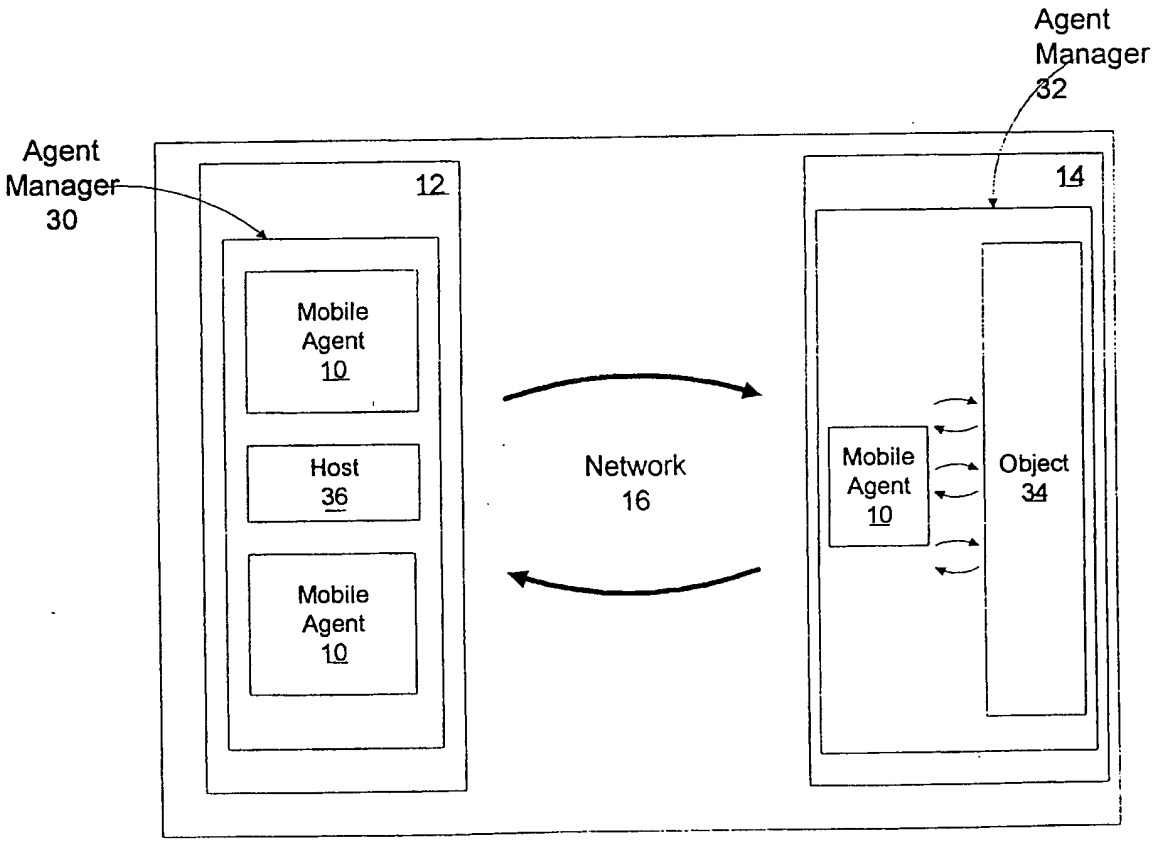

Agent Manager 32
Agent Manager 30
12
14
Mobile Agent 10
Host 36
Mobile Agent 10
Network 16
Mobile Agent 10
Object 34

Fig. 4

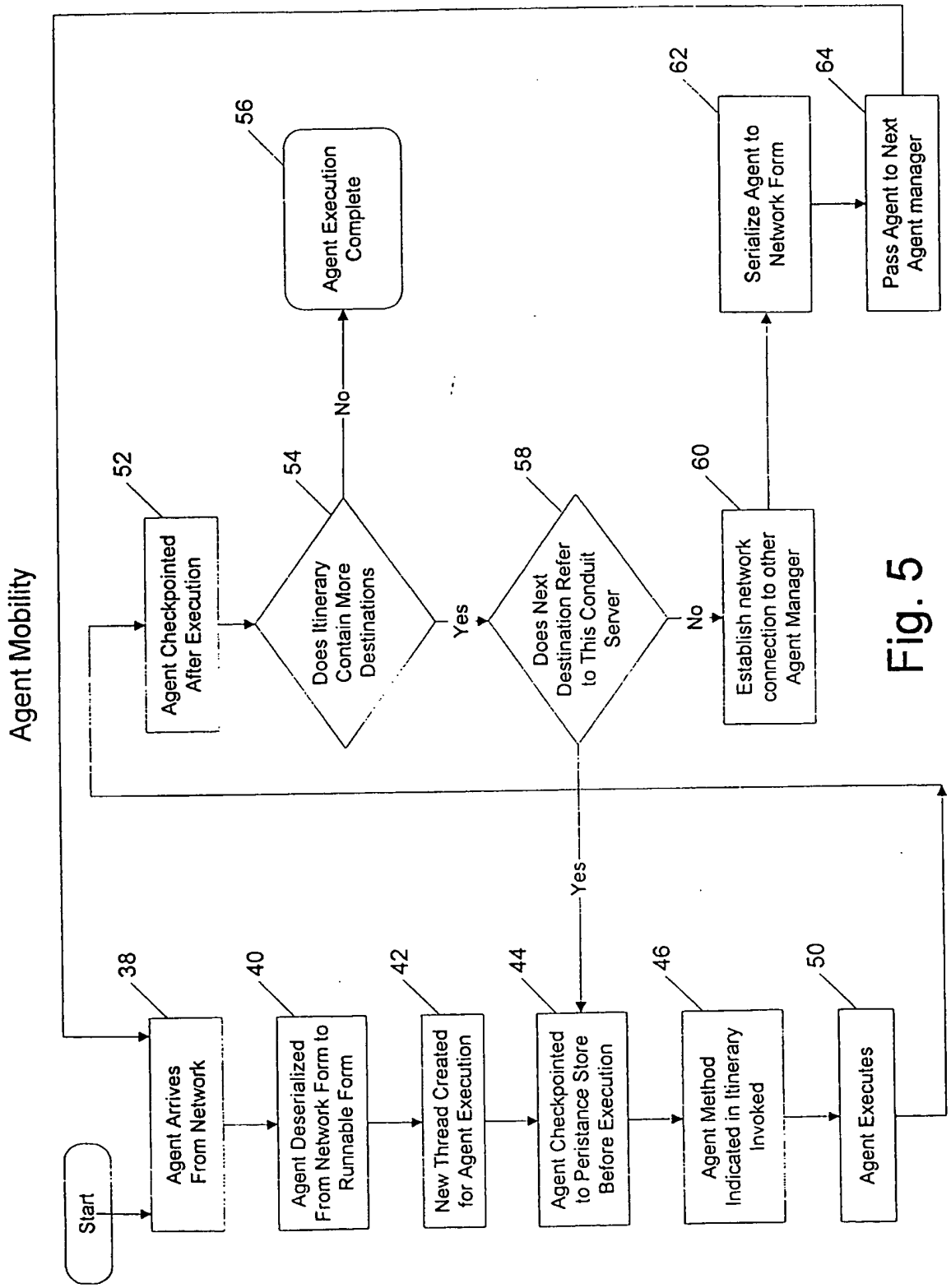

Agent Mobility
Start
Agent Arrives From Network
38
Agent Deserialized From Network Form to Runnable Form
40
New Thread Created for Agent Execution
42
Agent Checkpointed to Peristance Store Before Execution
44
Agent Method Indicated in Itinerary Invoked
46
Agent Executes
50
Yes
Does Next Destination Refer to This Conduit Server
58
No
Establish network connection to other Agent Manager
60
Serialize Agent to Network Form
62
Pass Agent to Next Agent manager
64
Yes
Does Itinerary Contain More Destinations
54
No
Agent Execution Complete
56
Agent Checkpointed After Execution
52

Fig. 5

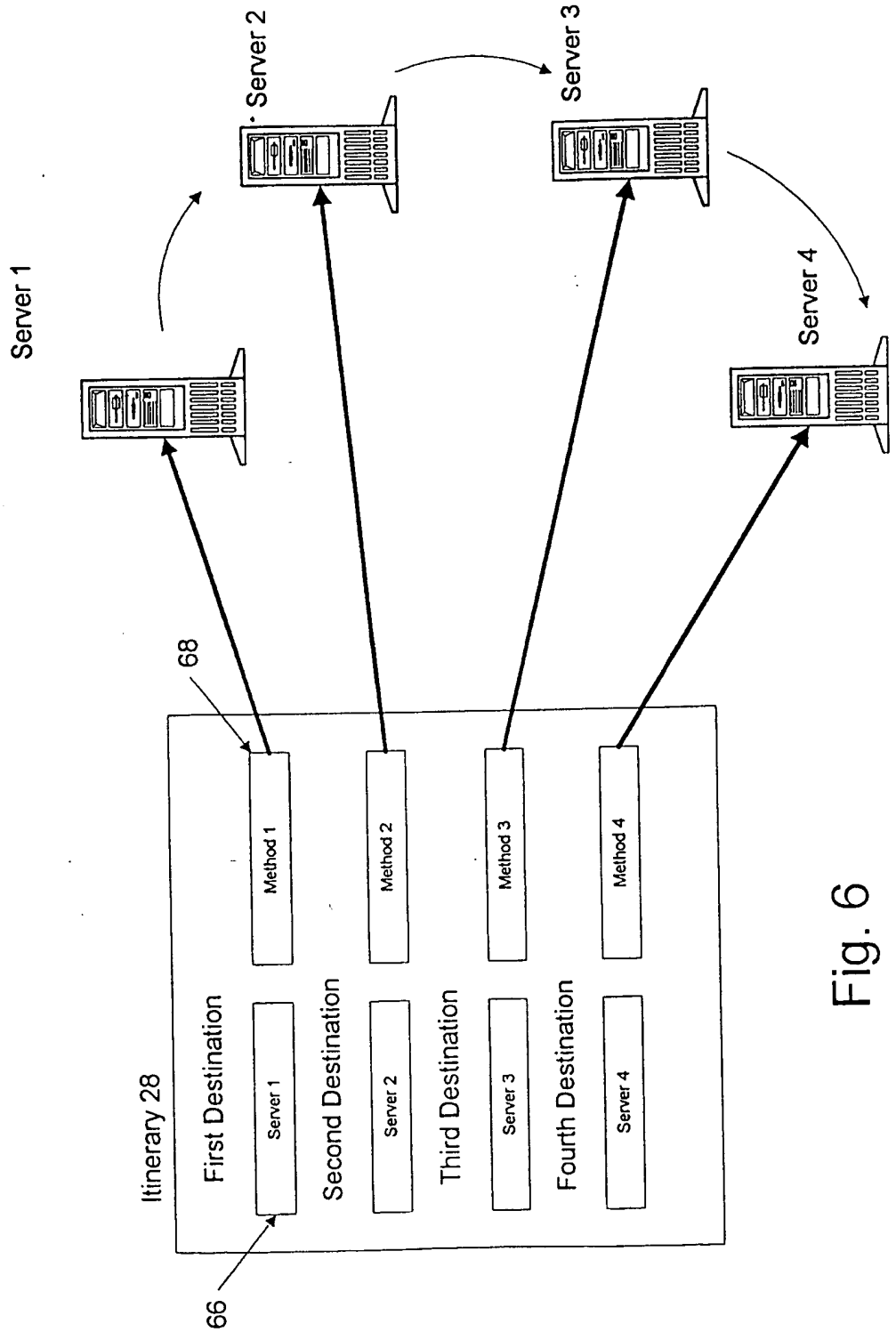

Server 1
Server 2
Server 3
Server 4
68
66
Itinerary 28
First Destination
Second Destination
Third Destination
Fourth Destination
Method 1
Method 2
Method 3
Method 4
Server 1
Server 2
Server 3
Server 4

Fig. 6

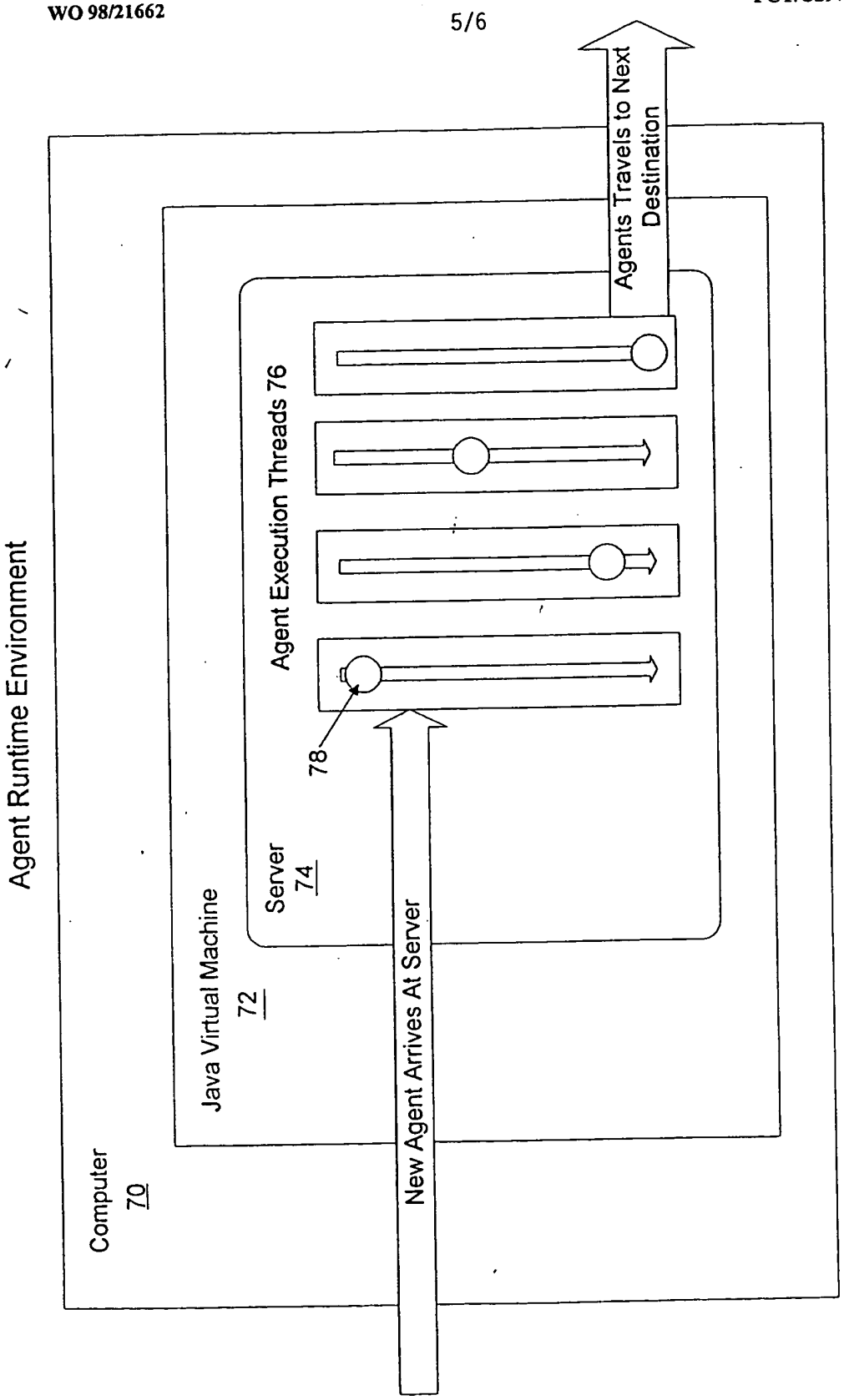
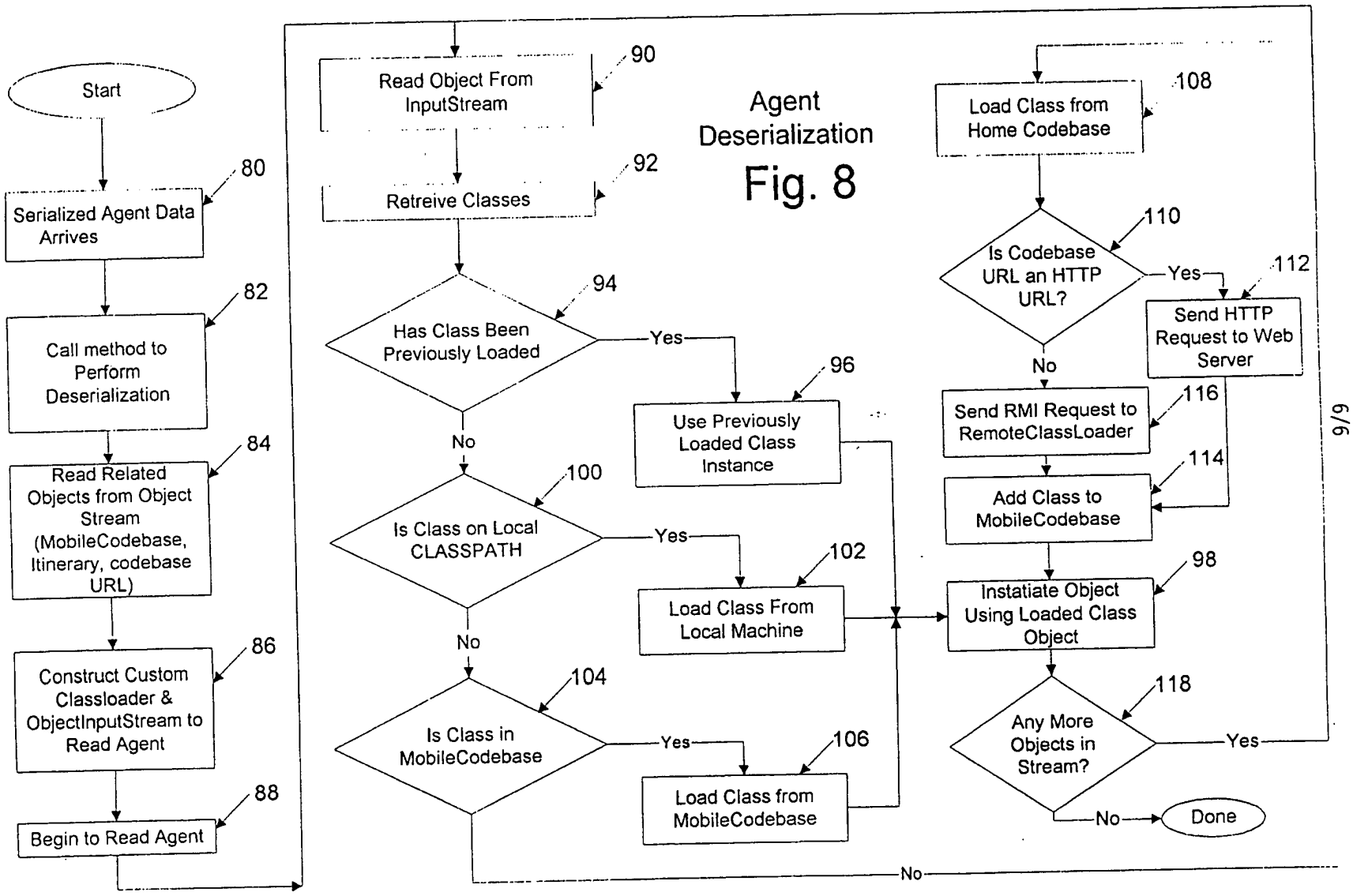

Agent Runtime Environment
Computer
70
Java Virtual Machine
72
Server
74
Agent Execution Threads 76
78
New Agent Arrives At Server
Agents Travels to Next Destination

Fig. 7


Start
Serialized Agent Data Arrives
80
Call method to Perform Deserialization
82
Read Related Objects from Object Stream (MobileCodebase, Itinerary, codebase URL)
84
Construct Custom Classloader & ObjectInputStream to Read Agent
86
Begin to Read Agent
88
Read Object From InputStream
90
Retreive Classes
92
Has Class Been Previously Loaded
94
Yes
No
Use Previously Loaded Class Instance
96
Is Class on Local CLASSPATH
100
Yes
No
Load Class From Local Machine
102
Is Class in MobileCodebase
104
Yes
No
Load Class from MobileCodebase
106
Load Class from Home Codebase
108
Is Codebase URL an HTTP URL?
110
Yes
No
Send HTTP Request to Web Server
112
Send RMI Request to RemoteClassLoader
116
Add Class to MobileCodebase
114
Instatiate Object Using Loaded Class Object
98
Any More Objects in Stream?
118
Yes
No
Done
Agent Deserialization
Fig. 8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/20232

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 13/00
US CL :395/200.32

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/200.32

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS, IEEE

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	BERGHOFF ET AL. Agent-based configuration management of distributed applications. IEEE. August 1996. Pages 52-59. Abstract and Introduction, pages 52-53; An infrastructure for mobile agents, pages 54-55; Agent-based configuration management, pages 55-57; and An agent-based management scenario, page 58.	1-46
Y	US 5,499,364 A (KLEIN ET AL) 12 March 1996, col. 1, lines 5-35; col. 3, lines 18-30; col. 4, lines 45-52; and col. 5, line 28 - col. 6, line 5.	1-46

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 21 MARCH 1998	Date of mailing of the international search report 13 APR 1998
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer LE HIEN LUU Telephone No. (703) 305-9650

Form PCT/ISA/210 (second sheet)(July 1992)*

THIS PAGE BLANK (USPTO)